AD-A115 560   AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DEVELOPMENT OF A MICROPROCESSOR-RASED ASYNCHRONOUS DATA COMMUNI--ETC(U)
DEC 81 D L HARTMANN
UNCLASSIFIED AFIT/GCS/EE/81D-11   NL

1 of 2
AD
A115560

AFIT/GCS/EE/81D-11

DEVELOPMENT OF A MICROPROCESSOR-BASED ASYNCHRONOUS DATA COMMUNICATIONS LINE TESTER

THESIS

AFIT/GCS/EE/81D-11 David L. Hartmann
Capt USAF

Approved for public release; distribution unlimited.

AFIT/GCS/EE/81D-11

DEVELOPMENT OF A MICROPROCESSOR-BASED ASYNCHRONOUS DATA COMMUNICATIONS LINE TESTER

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

David L. Hartmann, B.S.
Capt USAF

Graduate Computer Systems
December 1981

Approved for public release; distribution unlimited.

Preface

This thesis delved into the world of microprocessors. Even though microprocessors have major limitatations in capabilities, it is the overwhelming success that they have in the special arenas that make them shine. An emerging field of microprocessor utilization is data communications. The field of communications line testing is critical to all users who must be involved with leasing communications lines. Testing is expensive in terms of down time and delays. It is for this reason that I chose this project to be able to provide a needed product.

I would like to express my gratitude to my thesis advisor Dr. Gary Lamont for guiding me through the project and to Ltc James Rutledge and Maj Walter Seward for providing those much needed comments. In addition, my sponsor Carlos Fernandez of the Weapons Laboratory was very helpful in providing the necessary equipment and information I needed to complete my work and I would like to express my thanks to him.

The demanding nature of any thesis project is hard on the student accomplishing the thesis, but it is equally as hard on the spouse of the student. I am indebted to my wife for her constant support throughout this project.

David L. Hartmann

Contents

Page

Preface ii

List of Figures v

List of Tables vi

Abstract vii

I. Introduction 1

Background 2
Objective 4
Approach 5
Organization of Thesis 7

II. Requirements Definition 9

Introduction 9
Context Analysis 9
General Requirements 12

Software Modularity Requirement 12
Software Maintainability Requirement 13

Functional Requirements 16

Man-Machine Interface Requirement 16
Real-Time Logic Requirement 17
Self-Loading Software Requirement 17
Monitor Mode Requirement 17
Simulate Mode Requirement 21

Design Constraints 26
Summary 27

III. System Design 28

Introduction 28
Software Environment 29
Software Design Techniques/Strategies Used . . . 30

Transaction Centered Design 31
Structure Charts 32
Coupling 32
Cohesion 33

Line Tester Functions 34

The Driver Function 35
Real-Time Logic Funcion 36

Man-Machine Interface Function 39
Monitor Mode Function 41
Simulate Mode Function 46
The Line Parameter Function 52
The Utilities Function 55

Summary 58

IV. System Implementation and Testing 60

Introduction 60
Implementation 61
Testing 65
Summary 69

V. Recommendations and Conclusions 70

Introduction 70
Recommendations 70
Conclusions 71

Bibliography 74

Appendix A: Data Flow Diagrams 77

Appendix B: Mostek Z-80 Hardware Components 85

Appendix C: Structure Charts 90

Appendix D: Line Tester Users Manual 152

Appendix E: Programming The SIO 170

Vita 175

List of Figures

Figure Page

1 Data Flow Diagram (DFD) Components 11

2 System Diagram--Perform Data Line Test 14

3 Execute Timed Test (2.0) DFD 15

4 Enter Monitor Mode (2.3) DFD 19

5 Perform Monitor Mode Test (2.3.4) DFD 20

6 Enter Simulate Mode (2.2) DFD 23

7 Perform Simulate Mode Test (2.2.4) DFD 24

8 Design of Main Driver for Line Tester 37

9 Submodule to Interpret Operator Input at System Level 38

10 Design of Monitor Mode Sub-Driver 43

11 Submodule to Interpret Operator Input at Monitor Mode Level 44

12 Design of Simulate Mode Sub-Driver 48

13 Submodule to Interpret Operator Input at Simulate Mode Level 49

14 Execute Simulate Mode Test 50

15 Design of Line Parameter Sub-Driver 53

16 Overview of Module Interfaces 59

17 Physical Cable Connections for Monitor Mode 66

18 Pin Configuration for Cable Setup 67

List of Tables

Table Page

I. Simulate Mode Test Bit Patterns 47

II. Mostek Z-80 Memory Configuration 62

III. Quick Reference of Commands 64

Abstract

Software was developed to support a microprocessor-based asynchronous data communications line tester which operated on digital data. The software functions included a monitor mode and a simulate mode both of which could be executed through interactive dialogue with the operator of the system. The software was to reside in PROM and execute upon power up and reset.

A structured top down approach was used to design a transaction oriented software line tester. The design was implemented on a Mostek MDX microcomputer system which included a serial input/output (SIO) interface, a Z-80 CPU module, 32K dynamic RAM module, and a EPROM/UART module. Data line testing was performed through the SIO utilizing its interrupt mode and status gathering capabilities to capture error conditions. The software, written in Z-80 assembly language, was tested using both "white box" and "black box" testing strategies.

The microprocessor-based line tester proved to be versatile and efficient in performing asynchronous data communication line testing. Designing the software using the top down approach was effective in permitting structure changes. The large and powerful Z-80 instruction set provided a great degree of versatility for performing the line tester functions.

DEVELOPMENT OF A MICROPROCESSOR-BASED ASYNCHRONOUS DATA COMMUNICATONS LINE TESTER

I. Introduction

The utilization of computers to automate tasks varies from large scale computing facilities which could fill an entire room to very small single printed circuit board computers for specialized tasks. An increase in computer utilization has come about because of the advances in computer technology which have reduced the cost and size of computers without a detrimental reduction of capabilities.

The very small computers, such as the mini and micro computers have experienced the greatest increase in utilization because of their increased capabilities and low cost. In particular, the microcomputer, which has as its main component the microprocessor, has been used widely by the U.S. Air Force in specialized environments (Ref 1).

Microprocessors have found a niche in a vast number of applications. It is inevitable that microprocessors carve a niche in data communications. The flexibility and versatility of microprocessors make them a valuable device for use in communications. Considering the processing capabilities and with the addition of memory and control elements, microprocessors can provide enormous computing capacity. Yet, they consume little power and are inexpensive. Microprocessors have performed best in a limited applications environment where each system performs specialized functions. Such a limited application area is the topic of this thesis where a microprocessor is used in testing of asynchronous data communications lines.

Background

Data communications between a host computer and the terminals connected to it is expected to be reliable. When transmitting digital data, errors can change the meanings of instructions or alter the value of data. Various error detection schemes can be employed. These schemes are highly application dependent but usually involve such schemes as parity encoding and checksums (Refs 3;8;18). Clearly then, the support and maintenance of these data communications lines requires versatile diagnostic tools. These diagnostic tools should also indicate how often an error has occurred to ascertain the reliability of the transmission line. In addition, a greater demand for data communication between host computer and terminal has led to a need for better and more flexible test equipment. Testing will help determine whether it is the transmission line, the computer or the terminal which is at fault if a problem exists. A microcomputer has been chosen to base the line tester because it provides the versatility desired. One of the reasons microcomputers are versatile is because they can be programmed and thus changed to adapt to new environments or perform new functions.

The Air Force Weapons Laboratory (AFWL) is experiencing support and maintenance problems on asynchronous data communications lines. Consequently, the Weapons Laboratory offered to sponsor the development of a microprocessor-based asynchronous data communications line tester. Such a device would be connected to the data line between the modem and the terminal or host computer. Connection of the tester here involves the collection of digital data versus analog data.

The concept behind developing a digital oriented line tester versus an analog line tester is based on versatility. Digital error detection is actually more inclusive by allowing the user to do much of his own testing through the modem without the need for the complex test equipment used for analog data testing (Ref 6). With digital error detection using the microcomputer, many line parameters may be varied in performing the tests on the communications line. Parameters such as baud rate, bits per character, asynchronous and synchronous control characters, parity, etc., can be controlled via the microcomputer and the input/output interface hardware.

The hardware components for the data line tester such as the microprocessor, memory, console interface, and data line interface are off-the-shelf equipment. The acquisition of existing off-the-shelf hardware is quicker and less costly than custom made systems and provides for flexibility. The Weapons Laboratory has already supplied these components to test the software to be developed. The hardware is described fully in Appendix B.

Objective

The problem addressed in this thesis investigation is the development of structured software to support a microprocessor-based asynchronous data communications line tester.

The software for the line tester is to be developed to perform the testing functions outlined by the Weapons Laboratory which are more fully described later on in this report. The line tester functions required by the Weapons Laboratory designate that the software will be designed to operate in two modes: monitor mode and simulate mode. Operating in the monitor mode is defined as the ability to receive data. This data will then be used to produce traffic statistics, error conditions as a function of time, and histograms of line performance for a given time period. Operating in the simulate mode is defined as the ability to receive and transmit data and perform tests. These tests shall include loop back tests using standard messages and other bit oriented pattern tests.

Another major objective along with the development of the two previous modes of operation is the development of the man-machine interface software to provide the versatility in the selection of testing functions and control of testing parameters.

This project is intended to provide a basis for the development of a more comprehensive data communications line tester. The Weapons Laboratory has plans to expand the functions currently desired of the line tester software. Accordingly, the structure of the line tester software will be designed to simplify the modification task.

A limiting factor regarding the statistics gathered by the line tester software involves the hardware to be used. The accuracy of the statistics gathered is directly related to the capability of the data line interface hardware which monitors the communications line, namely the Mostek Z-80 serial input/output (SIO) chip. No attempt will be made at an analysis of transmission errors by using the analog input of the communications line.

Approach

The development of software requires that the developer look at the software's present and future intended functions. It turns out that the development of the software is the easiest part while maintenance and modifications to software are the most costly and time consuming (Ref 17). Since the major effort in this thesis investigation is software development, the original design is to have a structure which is amenable to the maintenance and modification task. An original design constructed this way prolongs the software's usefulness and versatility and follows good software engineering principles.

The approach taken to develop the tester software was an iterative process. The restructuring and modifying of code in the lower levels sometimes affected the structure of the upper levels. A cyclic process resulted, and the final outcome is what is presented in this thesis investigation. The iterative process used is discussed next.

The initial phase of this project consisted of a literature search for information on asynchronous data communications line testing, simulation, and structured program design. Since the software must be compatible with the hardware on which it is to run, a detailed study of the hardware was performed in this phase, also. Additionally, a requirements analysis was performed to establish a basis for the design of the line tester. Data flow diagrams were used to graphically represent the requirements of the data communications line tester. The data flow diagrams helped to define the necessary functions of the line tester by depicting the logical flow of data through the system (Ref 20:25). A clear, concise definition of the functional requirements resulted. The functional requirements, depicted by the data flow diagrams, are detailed in chapter II.

As will be indicated in the requirements definition section, the software must be modular and maintainable. Therefore, in generating the software for this project a top down structured design approach was used (Ref 21:3). In this approach, the software was designed by hierarchical levels, with the top level as the problem of this thesis broken down into major functions. Subordinate levels are further breakdowns of the major functions of the top level. The structured design approach provides the user of the software with a concise, understandable description of the system which is necessary for future maintenance and modifications.

Structure charts were used to graphically depict the modular construction of the software. Not only do the structure charts provide a time independent model of the hierarchical relationship of the modules within a program, but also are an extremely powerful tool in considering the consequences of structure changes by displaying the nature of what is changing (Ref 21:42).

The final phase of this effort was the implementation and testing of the line tester software. In this phase the software was validated against the functional requirements of the previous section. Testing of the software was performed to locate errors in the program and ensure a reliable product.

Organization of Thesis

This thesis will cover the development of a microprocessor-based asynchronous data communications line tester by stepping through fundamental procedures of a development process. Initially, and as already presented above, chapter I is an introduction to the thesis. It provides the background, objective, and approach for the thesis. Chapter II will describe the requirements of the data communications line tester, or in other words, what the system must accomplish. Chapter III will specify the design of the software for the line tester. Chapter IV will specify the implementation and testing procedures and results. Chapter V will conclude the thesis with recommendations and conclusions.

II. Requirements Definition

Introduction

In order to lay the groundwork for the next stages in the development of the software for the line tester, a thorough analysis of the needs which it is to fulfill is necessary. This phase will be the requirements definition phase. Provided in the requirements definition phase are three main sections (Ref 19). The context analysis section will specify the reasons the line tester is to be developed. Secondly, the functional requirements section will describe the functions which the line tester must accomplish. Finally, the desijn constraints section will describe any limitations imposed upon the design of the line tester software.

Context Analysis

The microprocessor-based data communications line tester will serve as a diagnostic tool for the maintenance and support of asynchronous data communications lines. The need for this kind of diagnostic tool is based on a user requirement for maximum flexibility in a testing device. The flexibility of the microprocessor-based tester allows for additional enhancements which the Weapons Laboratory, as the user, has scheduled in the future plans for the line tester. Another reason for the use of a microprocessor-based line tester is to allow interactive access to the tester functions. This interactive capability satisfies the need to operate in two modes which are the simulate mode and the monitor mode.

A graphical description of the requirements is provided by using data flow diagrams. Figure 1 depicts the meanings of the symbols used with this tool. The data flow diagram provides a clear and descriptive form for defining the functions of the line tester. The data flow diagrams ignore initialization, termination, control loops and decisions, and model only the steady state behavior of the system (Ref 20:55). As will be seen later, the data flow diagrams suggest certain design strategies too.

The data flow diagrams (DFD) have four main components. They are the data flow name, process name, file name, and information sources and sinks. A data flow name represents data being passed between processes and is depicted by a curved line. The process is another component of the DFD which manipulates or converts input data flow to some desired output data flow. Processes are represented by the large circles. A third component of the DFD is the box. A box represents sources of information or sinks of responses. Sources and sinks can be user consoles, keyboards, or other input/output devices. A final component of the DFD is the file. A file is a repository of data and is represented by a straight line.

Data flow diagrams depict a hierarchy of levels. Starting with the system diagram, the processes within the

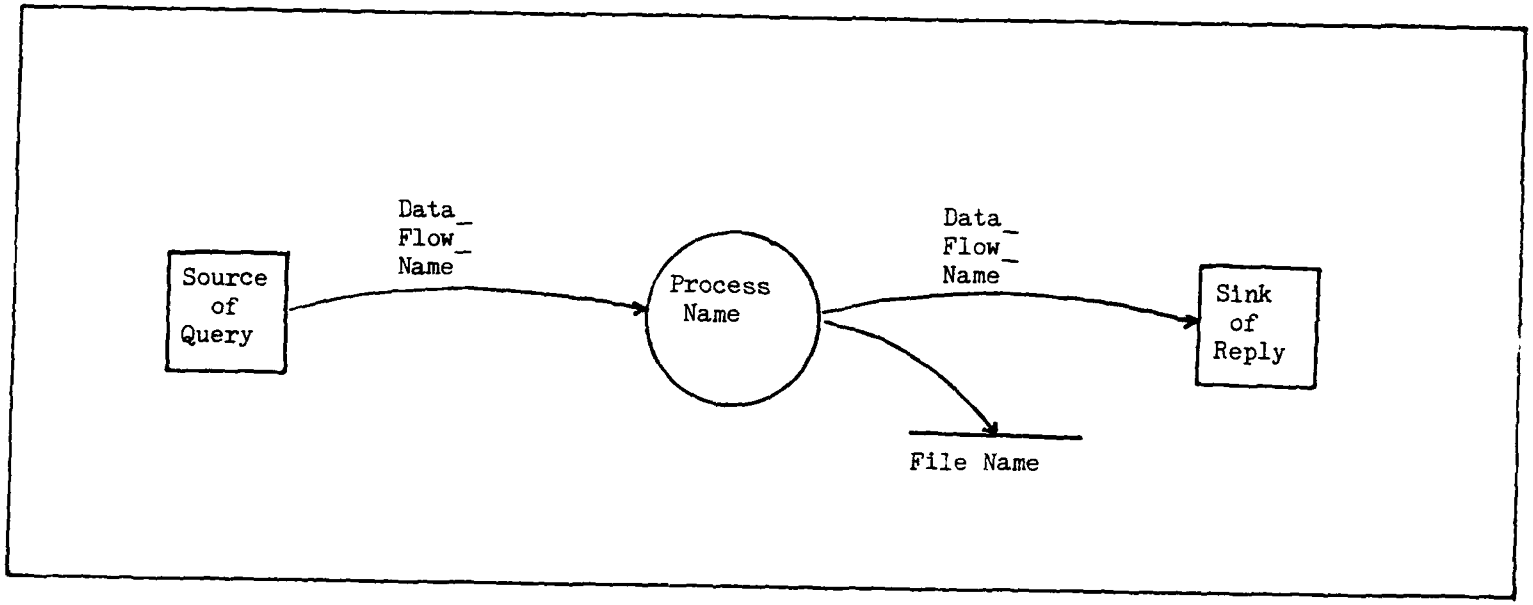


Figure 1. Data Flow Diagram (DFD) Components

diagram are expanded into lower level DFDs. The sub-dividing of processes continues until the data flows entering and leaving a process consist of only one data element.

The DFDs presented in this report depict the requirements of the line tester software at its highest levels only. The lower level requirements are discussed in this chapter but not depicted via DFDs.

General Requirements

Certain general requirements imposed by the Weapons Laboratory concern the structure and documentation of the software to be developed. These general requirements will be discussed in this section. The hardware requirement will not be addressed here because all necessary hardware components have been supplied. The capabilities of the hardware to satisfy the software functions are adequate (Refs 10;11;12;13;14;15). Appendix B describes the hardware components in detail. As will be indicated in the design chapter, the processing speed, memory capacity, and input/output interface of the hardware provides the software adequate bounds with which to perform its functions.

Software Modularity Requirement. The software to be developed must conform to the principles of top down structured programming. This is a user-oriented requirement imposed by the Weapons Laboratory. Software modularity is necessary to simplify the task of adding or deleting capabilities to the line tester.

Figures 2 and 3 depict the system design and the functional modules of the software to be developed. Each bubble in the figure represents one or more module thereby representing the hierarchical structure of the software. The modules are broken down to a level such that the lowest level module performs one function which has been defined in terms of its most elementary inputs and outputs. The software modules then will be pieces of the system. The modules will serve as building blocks to construct the entire software package.

Software Maintainability Requirement. The software to be developed must contain the necessary documentation and be of a modular structured design in order to simplify the maintenance task. Since the Weapons Laboratory will be maintaining the software after it is implemented, this is also a user-oriented requirement. Design of software with maintenance in mind is a necessary design principle. Maintenance of software has become more costly than development of software (Ref 17). This requirement is closely associated with the software modularity requirement.

An additional requirement is the preparation of a user's manual which will serve to document the operation of the line tester and to guide the user through a testing session. Figure 2 depicts the system diagram and also shows the man-machine concept from user input to user console.

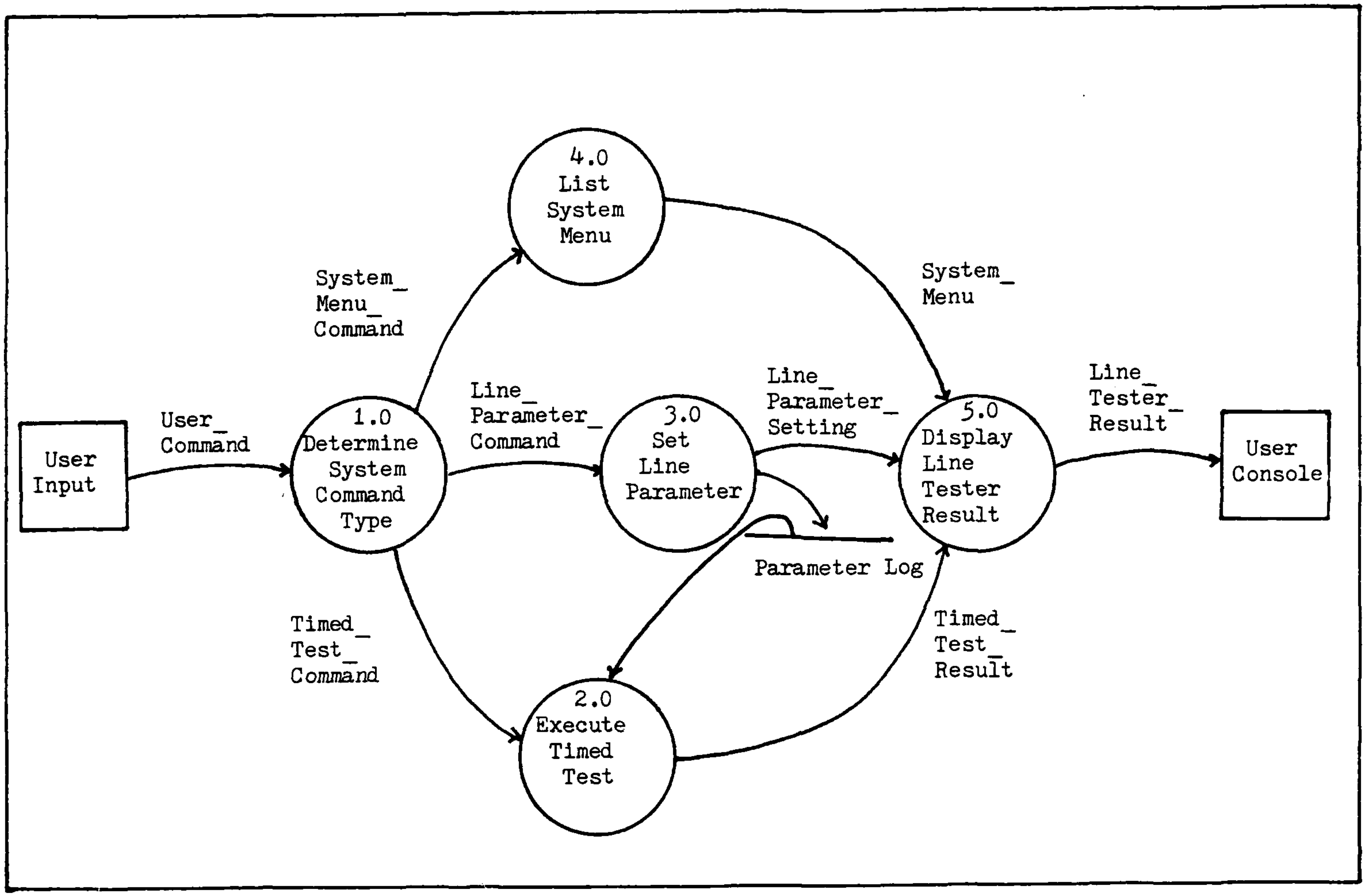


Figure 2. System Diagram--Perform Data Communications Line Test (Level 0)

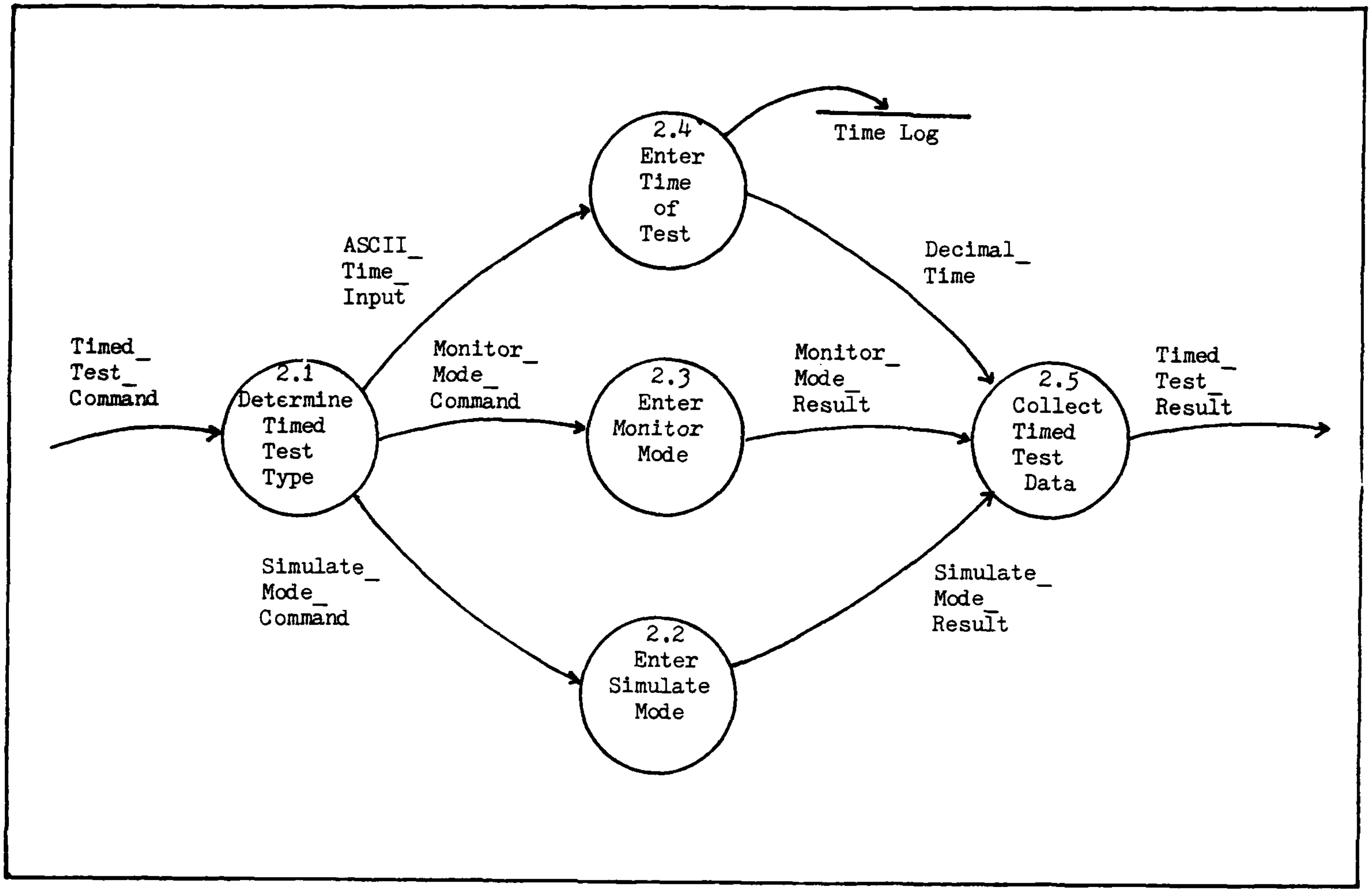


Figure 3. Execute Timed Test (2.0) DFD (Level 1)

Functional Requirements

The functional requirements of the microprocessor-based line tester define what functions the line tester must accomplish. The functional requirements deal mainly with the two modes of operation desired in the system. Figure 3 depicts the operational functions. Other functional requirements deal with interface requirements such as man-machine and real-time logic requirements.

Man-Machine Interface Requirement. In order to operate the microprocessor-based line tester in both a simulation and monitoring environment, a man-machine interface is needed. This requirement is necessary to give the line tester flexibility and changeability. The software to be developed here must provide the user with an interactive capability. The interaction software must provide the user with the necessary prompts and instructions to make the use of the line tester a relatively easy and understandable task. The amount of input required of the operator should be minimal to reduce the possibility of erroneous input. However, a great number of variables exist in the data communications environment that must be addressed to achieve proper communications (Ref 8). The operator therefore must be familiar with such line parameters as modem controls, baud rates, stop bits, parity settings, and bits per character settings. These line parameters will serve to establish the proper channel initialization for testing of the communications line.

Real-Time Logic Requirement. The microprocessor-based line tester must interface with the data communications line also. This requirement entails two aspects needed to perform the real-time logic function. The first is the processing of the incoming digital data over the data transmission medium being monitored or simulated. Efficient software is necessary to manipulate and process the incoming data and not lose any information. Information may be lost if processing of a received character has not completed before the next character arrives. The second aspect of the real-time logic function is timing the duration of a tester function. Software is needed to program a timer/counter to clock the duration of some of the test and to interrupt the tester upon timer runout.

Self-Loading Software Requirement. The software developed must be designed to execute upon power-up and reset since the software is to reside in programmable read only memory (PROM). This is a user-oriented requirement and is specified because there will be no mass storage device connected to the microcomputer system.

Monitor Mode Requirement. The monitor mode requirement is a line tester function requirement which is to be performed by receiving data through the serial input/output (SIO) device. The tests are to be performed and timed by the microprocessor for a relative time period set by the operator. The maximum relative time setting requested by the Weapons Laboratory will be 24 hours. Errors detected or characters received should be logged within the hour time interval in which they occurred. Both the transmit and receive lines of the transmission medium will be monitored to provide for complete testing of the communications line. The dual channel capabilities of the hardware serial input/output circuit permit the monitoring of both lines. The serial input/output circuit also is to be used to ascertain error conditions which may have occurred during transmission or reception of data. Figures 4 and 5 depict the requirements for the monitor mode. The monitor mode requirement consists of five tasks that must be accomplished.

The first monitor task requirement is to detect and log framing errors. A framing error occurs if a stop bit is not detected in its correct location after the parity bit (if used) or after the most significant data bit (if parity is not used). This task will be accomplished at the same time as the monitoring of parity errors. One error will be declared if both occur.

The second monitor task requirement is to detect and log the number of characters per unit time being sent over the transmission medium. This is a status gathering function. All characters received will be counted regardless if they contain errors.

The third monitor task requirement is to detect and log carrier loss errors. A carrier loss error occurs when there is a loss of the carrier signal between the terminal and the

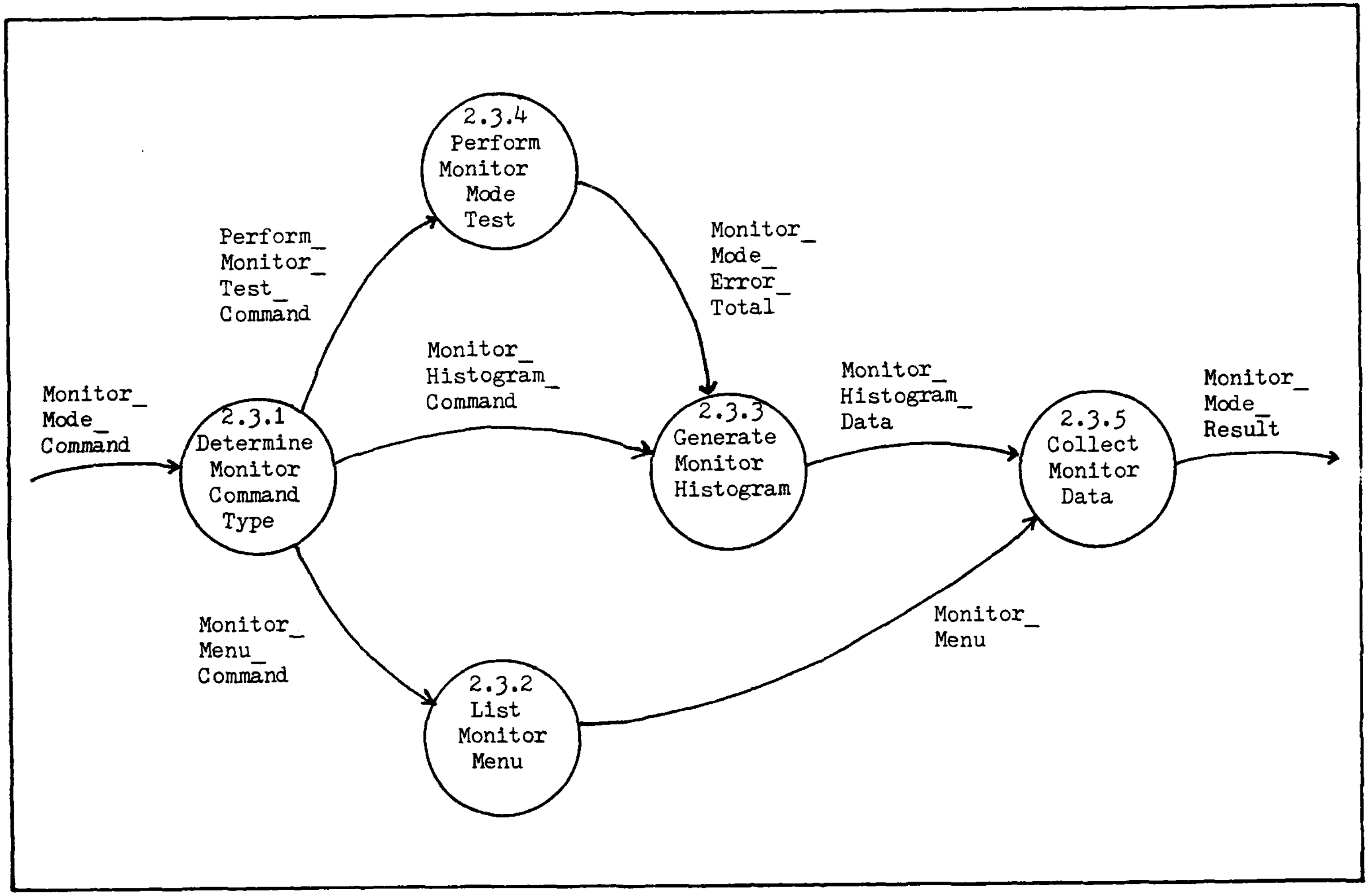


Figure 4. Enter Monitor Mode (2.3) DFD (Level 2)

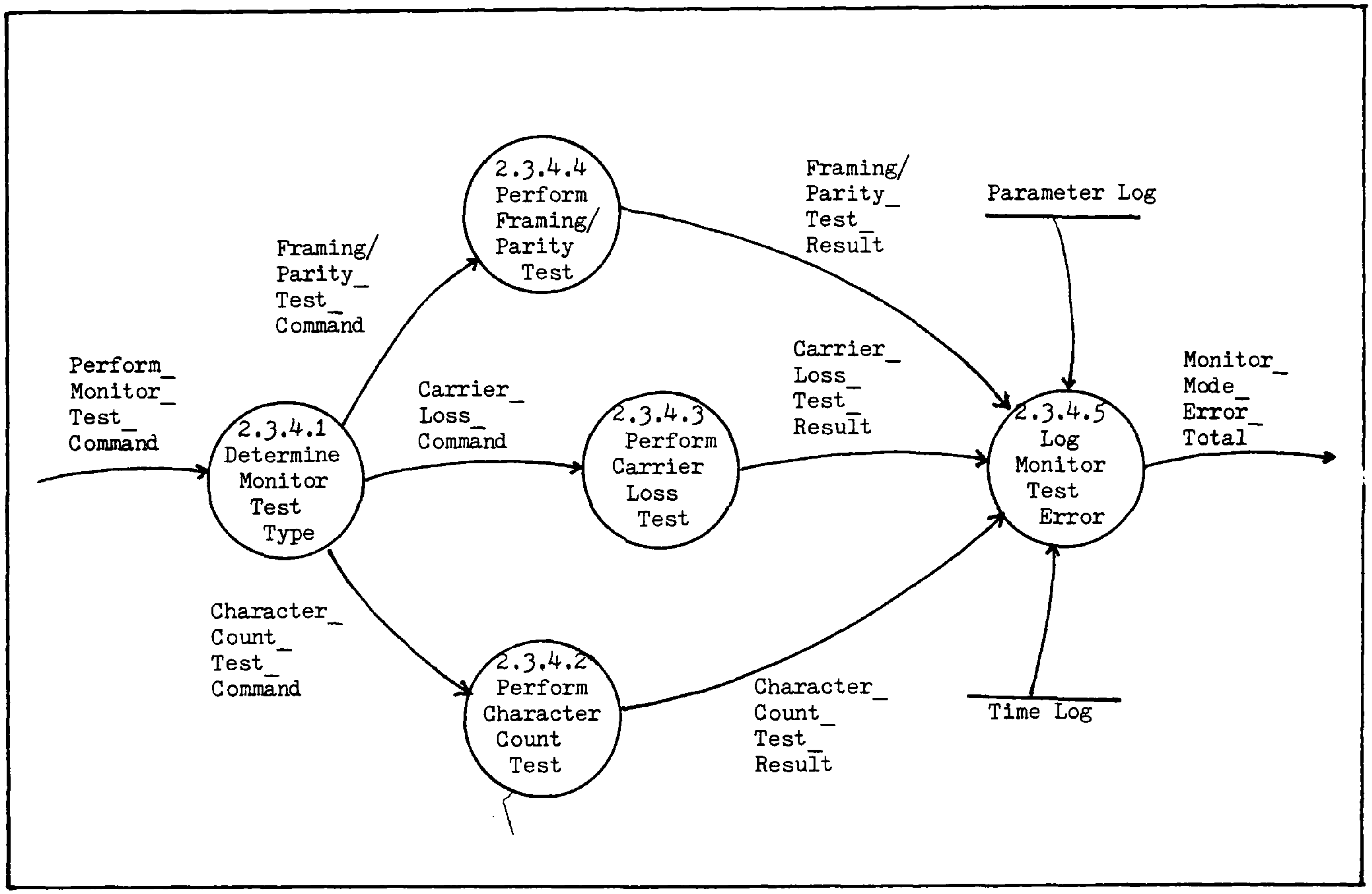


Figure 5. Perform Monitor Mode Test (2.3.4) DFD (Level 3)

host computer after communication has begun. The relative time input for measuring the duration of the loss is an operator input. Signal losses of duration greater than the period specified will not be counted as an error but considered as an end of message or transmission indication. Signal losses of duration less than the period specified will be counted as an error. The carrier signal flag of the SIO will be used to test the presence or absence of a carrier.

The fourth monitor task requirement is to detect and log parity errors. A parity error occurs when the parity bit of the character does not match true parity setting. The parity detection task will be performed in conjunction with the framing detection task as mentioned earlier.

The fifth monitor task requirement is to be able to display the monitor testing results in a histogram format on the user console. As requested by the Weapons Laboratory, the histogram should display the errors of a given type across the maximum relative time of 24 hours using one hour intervals. The errors of each type of test performed will be buffered to allow the operator to cycle through the data and display the results.

The monitor mode should display a list of monitor commands from which the operator can select.

Simulate Mode Requirement. The simulate mode requirement is a line tester function requirement which is to be performed by receiving and transmitting data through the SIO. The simulate tests are to be performed and timed by the microprocessor for a relative a time period set by the operator. The maximum time setting will be 24 hours. The Weapons Laboratory has requested that the transmission of data be a buffer at a time and continue for the specified time period. The Weapons Laboratory has requested the buffer size be 512 bytes in length. This size is large enough to send a meaningful amount of data and small enough to be accomodated within a smaller memory capacity system than the hardware configuration presently being used (Refer to Appendix B). The buffer of data will be transmitted through a loop back device and received and stored in a receive buffer at which time a comparison will be made between the characters sent and the characters received. In like fashion to the monitor mode, the errors detected in the simulate mode should be logged within the hour time interval in which the errors occurred. Figures 6 and 7 depict the requirements for the simulate mode. The simulate mode requirement consists of five tasks that must be accomplished.

The first simulate task requirement is to perform a bit error rate test. A bit error test is defined as the sending over the data line a series of characters which have bit patterns such as all ones or all zeros; receiving those same characters through a loop back device; and testing to see if what was sent matches what was received.

The second simulate task requirement is to perform a

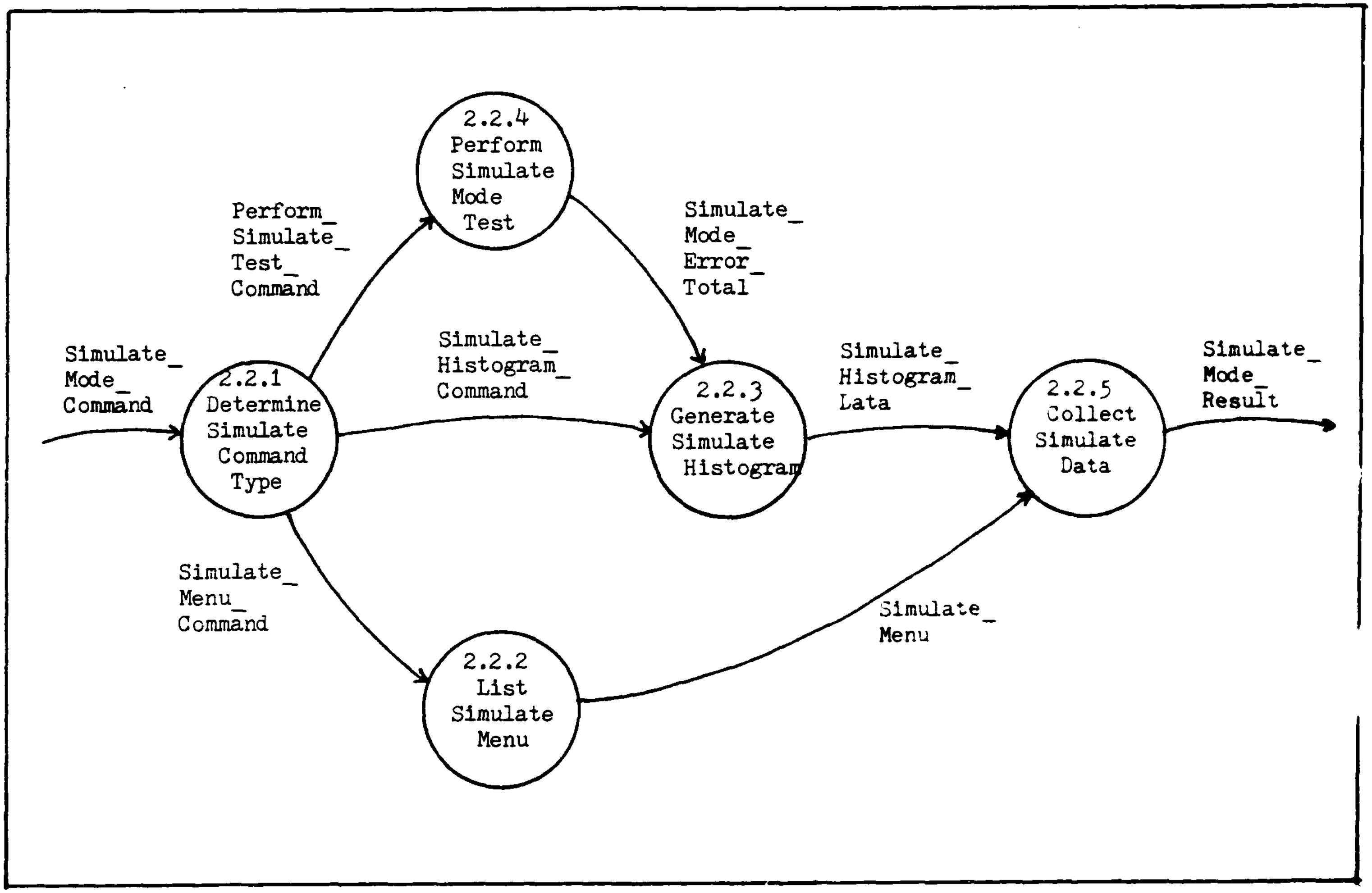


Figure 6. Enter Simulate Mode (2.2) DFD (Level 2)

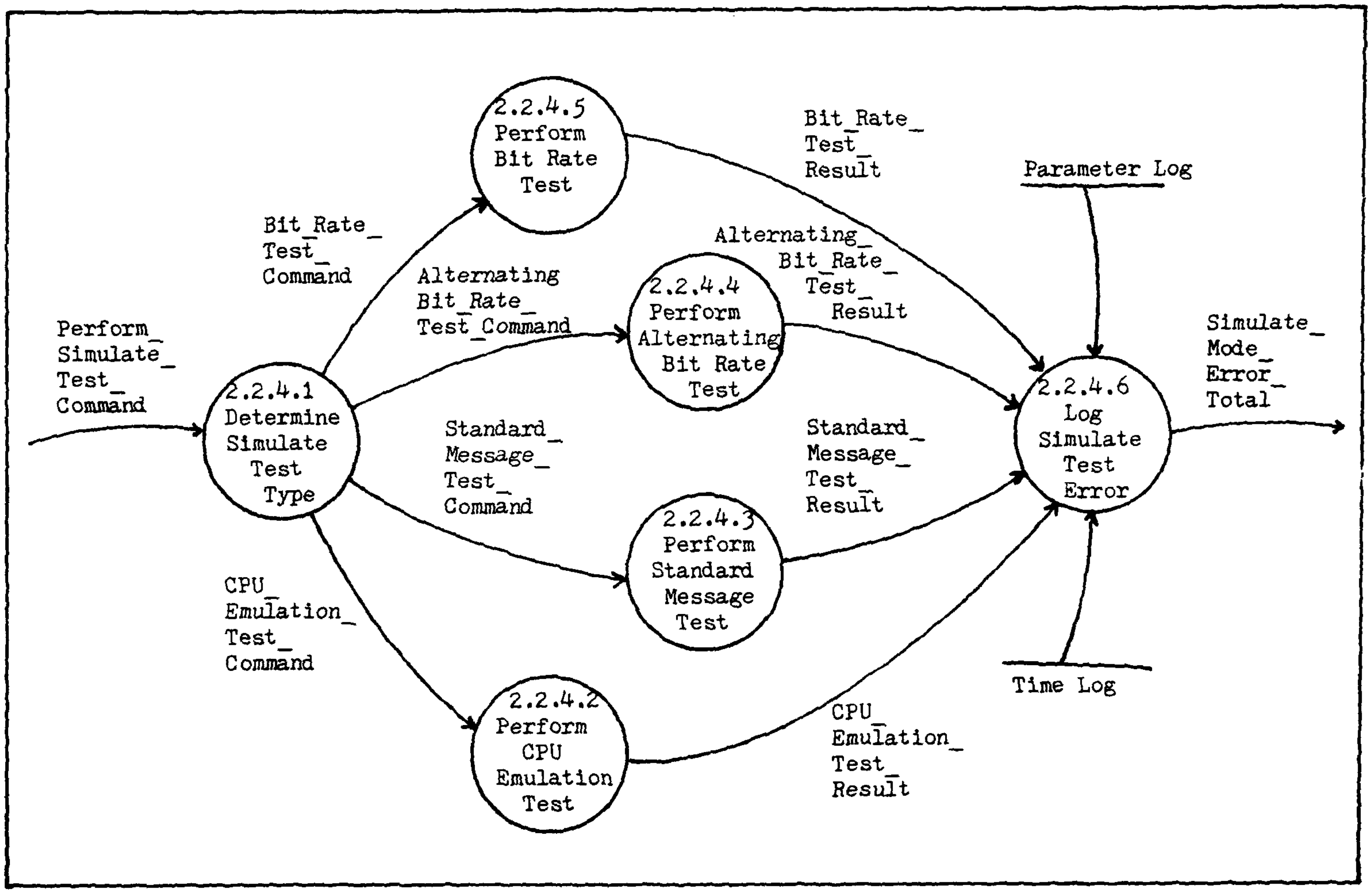


Figure 7. Perform Simulate Mode Test (2.2.4) DFD (Level 3)

standard message test. This task has two forms of input. In this test, a message, either generated by the operator or existing in the line tester software, is sent over the data line. The same message is received through a loop back device and displayed on the user console for visual inspection.

The third simulate task requirement is to perform an alternate ones and zeros test. This test is similar to the bit error rate test. In this test a pattern of alternating ones and zeros is sent over the data line. The data is received from the loop back device and matched with what was sent.

The fourth simulate task requirement is to perform central processing unit (CPU) emulation. In this test, the data input by the operator through the console is accepted by the line tester software and sent right back to the operator console. Validation of data input is achieved through this test.

The fifth simulate task requirement is to be able to display the errors found in the simulation tests in a histogram format. The histogram should be a display of the number of errors of a given type across a 24 hour base using one hour intervals. The errors of each type test will be buffered to allow the operator to cycle through the data and display the results of the tests.

The simulate mode should display a list of possible commands from which the operator can select.

In summary, the preceeding sections have covered the functional requirements of the line tester software. The functional requirements specified the functions to be performed by the line tester software. Indicated within these requirements are the tests which are to be developed. There will be three monitor mode tests which are depicted in Figure 5, and four simulate mode tests depicted in Figure 7. Appendix A contains the complete set of data flow diagrams used to describe these tests and other requirements.

Design Constraints

Much can be achieved through software if one has the hardware support. Hardware can be a limiting factor, but for this thesis investigation no profound limitations resulted from the hardware which was selected by the Weapons Laboratory. Appendix B has the complete list of hardware components supplied by the Weapons Laboratory, and a brief discussion of the hardware is given below.

The hardware provided by the Weapons Laboratory consists of four modules. The first module is the processor card with a Z-80 central processing unit, counter/timer chip, and 256 bytes of static random access memory (RAM). The processor operates on a 4 mHz clock. The second module is the serial input/output (SIO) card. The SIO has two full duplex channels with the multi-protocol capabilities. The status registers for error checking in the monitor mode are contained on this card. All line tester simulation and monitoring are performed using the SIO card. The third module is the memory card. This card has 32K bytes of dynamic RAM used for buffers and storage of line tester variables. The fourth module is the console serial input/output card. The console interface is achieved by this card which also has programmable read only memory sockets (PROM) for additional memory.

Summary

This chapter has defined what the microprocessor-based data communications line tester must accomplish. Each function of the tester was described and depicted graphically with data flow diagrams. The functions can be described as either satisfying a general or functional requirement. The general requirements were concerned with the software structure and documentation. The functional requirements were concerned with the actual tasks expected of the line tester. The complete system definition discussed in this chapter will be used to design the software using the top down structured approach.

software was to function. The design was initiated according to the results of this step.

Software Environment

In examining the software environment, several items were analyzed. These items included the language to be used, the implementation criteria, and the physical device on which the software was to function.

Since a major portion of the line tester tasks involve manipulations at the bit level versus blocks of characters, the language chosen was Z-80 assembly language (Refs 7;22). Assembly language also allows for coding of routines to be efficient by eliminating excessive overhead of the higher order languages. In addition, the Z-80 assembly language is quite compatible with the Z-80 central processing unit (CPU). The large variety of instructions available with the Z-80 code allowed for a more tailored selection of instructions to execute the functions of the line tester software.

Another item concerning the software environment deals with the implementation criteria. As specified in the requirements chapter, the software is to reside in PROM. Consequently all of the variables are located in one module for placement in RAM.

A final software environment item concerns the hardware. The line tester software will reside in a 'bare bones' system which will consist of only the processor, memory, and input/output interface. Consequently, the line tester software had to service all the hardware components; but, being a single processor system did not require the complex memory management, input/output device management, or processor management schemes of multiprocessor systems.

Software Design Techniques/Strategies Used

Structured software connotates levels in which drivers or controllers of software routines are the upper levels while the software routines which actually perform the task and are the basic elements of the routines are the lower levels. The design strategy is usually related to the level of software created first.

There are many strategies to apply to software design. One strategy is the bottom-up approach where the lowest level modules are created first. Another approach and the one chosen for this thesis investigation is the top down approach where the highest levels are created first. The choice of strategy revolves around the task to be solved and the preferences of the designer.

The design techniques chosen can be applied regardless of the strategy chosen (Refs 5;21). The software design techniques will aid in satisfying the software modularity requirements and provide continuity to the software design. The techniques employed are not all that are available but represent a firm basis to build software in the light of the requirements of this project. The techniques used include transaction centered design, structure charts, coupling and cohesion. These techniques will each be discussed now followed by a discussion of the line tester functions.

Transaction Centered Design. The modular design strategy of transaction analysis was used to generate the line tester software. The system is built around the concept of a transaction, that is, an element of data that triggers an action or sequence of actions (Ref 21:182). The data flow diagrams of the requirements definition chapter provide a graphical representation of this transaction centered concept. The structure depicted by the data flow diagrams strongly suggested the use of transaction analysis versus other strategies such as transform analysis, which is based on building a system around the concept of data transformation (Ref 21:171).

The functions to be performed by the line tester software depends upon the input of the operator. The operator may select from a displayed menu of items and the line tester software must interpret the input before performing any task. The selecting from a menu is therefore a transaction oriented process. Consequently, the line tester software was designed to be able to handle the input or transaction efficiently. Another major form of design, transform analysis, was also used in the line tester software in areas where only transformation of data from one form to another was required. The use of either or both strategies provided a guideline to follow in the design and will prove to be a valuable guide for future maintenance or modification tasks.

Structure Charts. The design tool used to depict the relationship between the modules of the software was the structure chart. The structure charts evolved after several iterations in the design of the line tester software. The structure charts help to show impacts upon other modules when new modules are added or when old modules are deleted. In addition, the structure charts will serve as a helpful guide to any future software modifications which may occur. Appendix C contains the complete set of structure charts used in this design project.

The components of the structure charts are simple. The boxes will represent a process or function to be performed. Lines connecting boxes will represent relationships between the processes, usually depicting the elementary tasks needed to perform that function. Arrows with open circles will represent data being transferred between processes while arrows with closed circles will represent control information. If the line has no arrow, then no data is being passed. A line with no data transfer is normally a subroutine call with no argument.

Coupling. Implied in the structure charts is the coupling of the modules. Coupling is defined as the measure of strength of association or interconnection between modules (Ref 21:187). To support the modularity requirement, the line tester software was designed to have loosely coupled modules which means that the modules are as independent of each other as possible. The value of loosely coupled modules is realized ir the testing and debugging phase of software design. Since the modules are relatively independent, modifications in one module can be made without having to know much about the functions of the other modules.

However, communications must exist between modules. An effort was made in the design of the line tester software to keep the communication between modules clear and obvious.

Cohesion. Also implied in the structure charts within the modules themselves is module cohesion. Cohesion is defined as the degree to which the elements within a given module relate to the accomplishment of a simple identifiable task (Ref 21:191). An effort was made to achieve a high level of cohesion within the modules. Some module elements are related by contributing to the execution of only one function while others are related because the module elements transform the same input data. Still other modules have related elements because they execute a sequence of steps to output a desired result. Any of these forms provide for a high level of cohesion by making the elements of the module strongly related.

Module cohesion is also affected by flags and decision points. In fact, there are many flags and decision points in a transaction centered design such as the line tester software. The excess use of flags and decision points tends to make software in general hard to follow. This is especially true when others have to maintain or modify your software which is the case here in this thesis investigation. Accordingly, the line tester software has a minimum amount of flags and decision points. The structure charts aided in the design of the line tester software by depicting the location of decision points and flags. The elimination of excessive decision points and flags reduced the complexity and increased the efficiency of the software.

Line Tester Functions

Structuring of functions of the line tester software is modular as prescribed by the requirements of the previous chapter. Not all functions will be discussed in this section but the main ones will be covered. The details of each function or the modules which perform that function can be found in the structure charts in Appendix C, and the code which is a separate document obtainable through the Air Force Institute of Technology (AFIT) School of Engineering or from the Computation Division of Air Force Weapons Laboratory.

The line tester software has been designed by modules to divide the functions into manageable pieces. By concentrating on one function, the software comprising that function can be kept relatively simple and workable. In covering the main functions, several points will be brought out. The areas to be covered include what the task is, how the task was accomplished, and justification on why it was designed that way. Since the functions are performed by the software modules and hardware, it will be necessary to discuss the structure of the modules also. The modules will only be discussed briefly so a more detailed description of any of the modules may be obtained by referring to the Appendicies. The modules have been constructed using the software techniques previously covered.

The line tester software has seven main functions. The main functions include the driver, real-time logic, man-machine interface, monitor mode, simulate mode, and implied in the preceeding two modes are the line parameter and utilities functions. These functions represent the main thrust of this effort as previously defined by the requirements chapter.

In the next sections, references will be made to the main modules of the software, and the names of modules will appear in capital letters. Two modules which will not be discussed are the DATA and MSG modules. They contain the RAM variables and the output messages respectively. Discussion of the line tester functions and modules will begin with the driver function.

The Driver Function. The operator must have some control over the line tester functions. This control is provided by the driver function by allowing the operator to select the test to be performed. The driver function is achieved by module MAIN. The function of MAIN is to control the line tester functions at a system level. The structure charts of Figures 8 and 9 depict the upper level design of the MAIN module functions. The components of the stucture charts depicted within Figures 8 and 9 and throughout this chapter were discussed previously within the structure charts section.

The driver function also initializes the variables which reside in random access memory (RAM), the console interface hardware, and the interrupt vectors. The initialization of these variables and control of entry points into the line tester software by the driver function also fulfills the self-loading requirement.

The design of a main driver follows from the top down structured approach of software design. All tasks which are to be performed are initiated and controlled by a driver. In this manner a known starting point is established which aids in debugging.

Real-Time Logic Function. The real-time logic function is performed within the monitor, simulate, and utilities functions. This function is discussed separately to point out its importance in the overall performance of the line tester software. As specified in the requirements chapter, two real-time logic tasks are to be covered. The first task deals with character processing while the second task deals with relative timing.

The line tester software will be gathering data from an

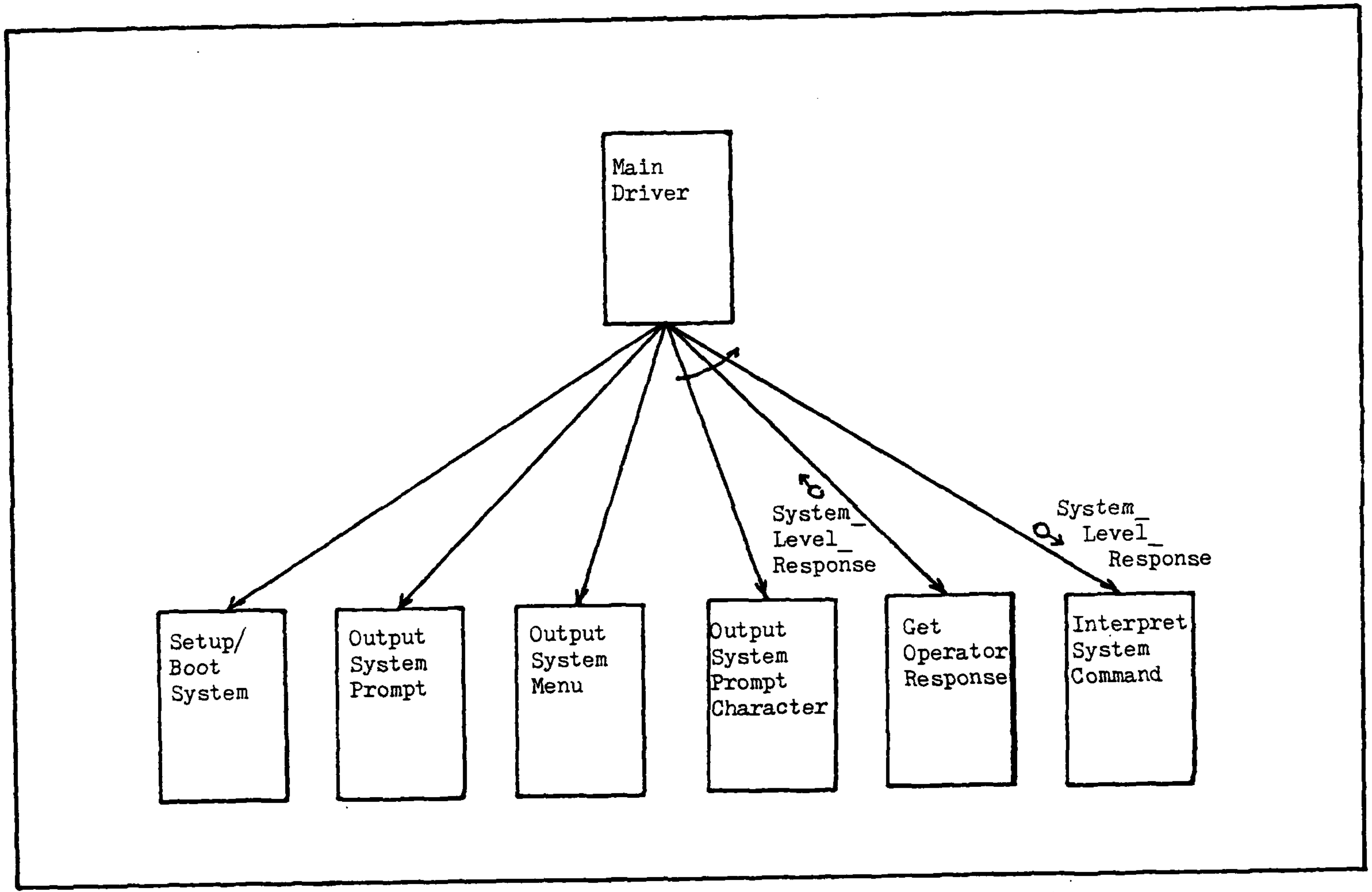


Figure 8. Design of Main Driver for Line Tester (Level 0)

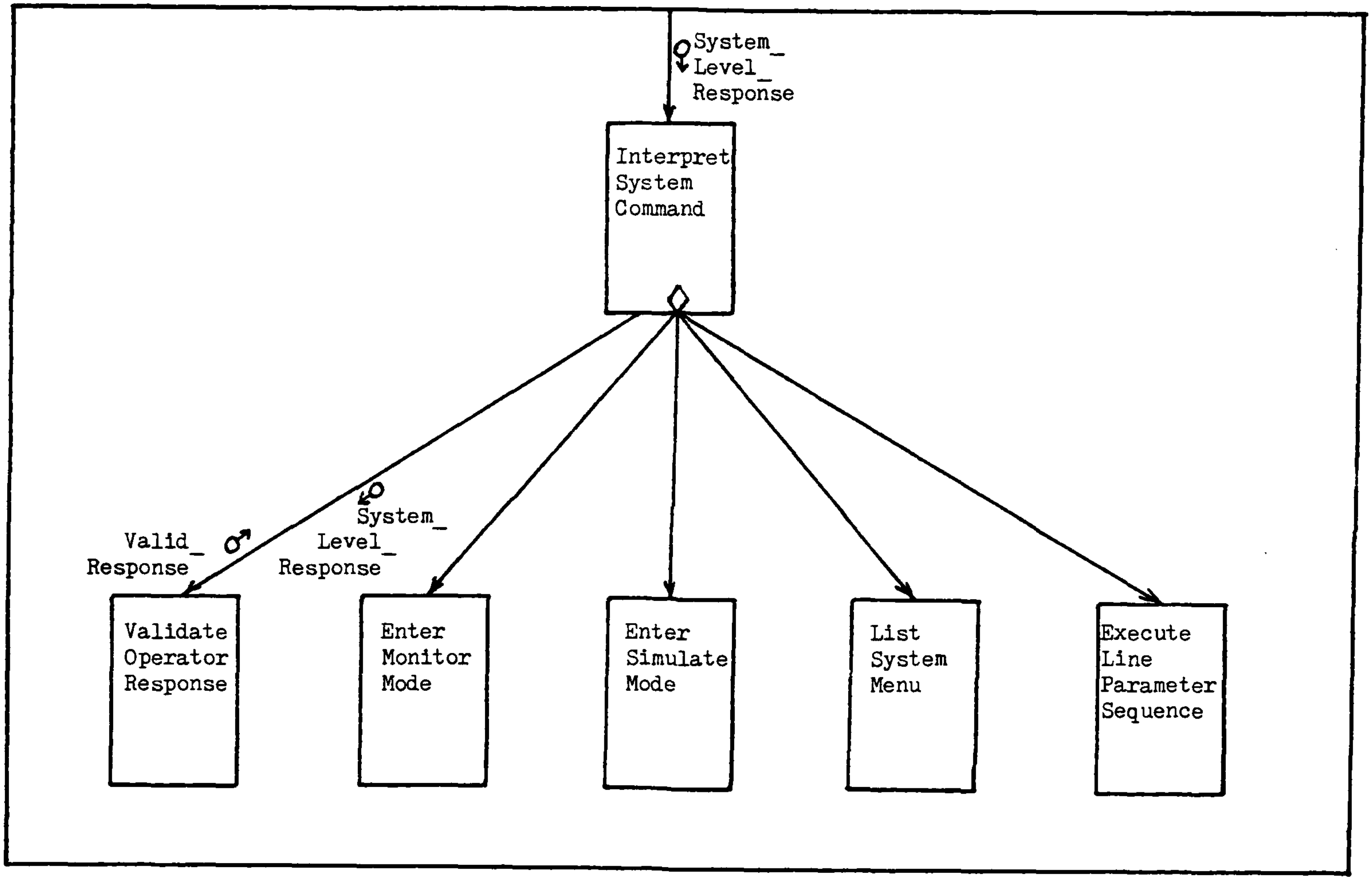


Figure 9. Submodule to Interpret Operator Input at System Level (Level 1)

active communications line. Any processing of statuses accompanying that data must be fast enough so as to be finished and ready to process the next data from the communications line. The processing speed of the central processing unit (CPU) of 4 mHz allows for the execution of approximately 100 Z-80 code instructions between characters when receiving data at a baud rate of 19,200 bits per second (Ref 15). The 19,200 baud rate is the fastest rate which will be used by the line tester. The critical code within the line tester module which handles character processing is well under 100 Z-80 code instructions. Therefore, the combination of CPU speed and efficient software permits this first real-time function.

Another real-time function is the clocking of the monitor and simulate mode test. Relative timing is needed to flag the end of a test. Relative timing is achieved by utilizing the counter/timer chip (CTC) on the CPU card. The CTC has four channels. Two of the channels can be cascaded (hardwired) to form the main timer. A third channel can be used as a secondary timer for the monitor mode carrier loss test. The timers can be programmed to increment by one second time intervals such that each interrupt is counted toward the relative time.

Man-Machine Interface Function. Interfacing with the operator is a critical step toward product useability. There are a variety of methods available to achieve man-machine dialogue (Ref 9). The method chosen for the line tester is computer initiated dialogue (Refs 8:75;9). In computer initiated dialogue, the computer leads the operator with prompting messages such that the operator need only respond with simple phrases. The responses can be commands as short as one character. Computer initiated dialogue reduces the amount of input required by the operator and thereby reduces the possibility of making an error on input.

The man-machine interface is achieved in large part by the console input/output (CONIO) module. The CONIO module provides console input and output capabilities for the line tester software. Submodules provide output of single characters to strings of characters. String output was provided to display menus and prompting information to the operator efficiently.

The structure of the CONIO module is straight forward. The CONIO software polls the universal asynchronous send receive transmitter (UART) for transmit or receive of data to or from the console respectively. The messages which are output to the console reside in this module.

The man-machine interface function is intermeshed with the other functions, and thereby interweaved within most of the modules. Locating the man-machine interface routines within the same module the routines support provides for greater module independence and aids in pinpointing errors. The CONIO module performs the actual input/output while other modules such as MAIN, LINE, MONITOR, and SIMULATE initiate the interface activity.

The inputs from the operator are validated to prevent an unwanted parameter to be used in the test performed. The man-machine interface software which performs the error checking resides in all the modules.

The Monitor Function. One major mode of operation is the monitor mode. In this mode of operation, the line tester receives through the SIO the same data that comes across the communications line between a host computer and a terminal. The SIO sets a status register associated with the character it received. Depending upon the type of test being performed in the monitor mode, the status register is examined and errors indicated (if any) are logged. The errors are logged within the hour interval they occurred as specified in the requirements chapter. The number of 8-bit bytes used for each hour interval is three. Three bytes allow for the accumulation of approximately 16.7 million characters or errors in one hour interval. The fastest baud rate to be used at present with the line tester software is 19.2 Baud. At 19.2 Baud, 8.6 million characters would be received in one hour. Therefore the 3-byte buffer for each h^our interval is sufficient for counting errors or characters received.

The monitor mode has its own sub-driver to control the different tests available. A sub-driver exists in the monitor module to allow the operator to repeatedly execute monitor tasks without having to jump to the system level.

The module which performs the monitor mode function is module MONITOR. Figures 10 and 11 depict the upper level design of the MONITOR module functions. The MONITOR module drives the different tests which may be performed in this mode. The monitor mode sub-driver reduces the complexity of the system level driver. The operator is prompted with a menu of possible commands upon entering the monitor mode. The operator can select from the list of commands by entering the appropriate character string which represents the task desired. The character strings are from one to five characters in length. Error checking is performed on the input from the operator

Before any task is performed in the monitor mode a check on channel initialization is performed. Both channels of the SIO must be active before a monitor mode task can be performed. Therefore, the operator must activate the channels and input the length of time each monitor mode test is to be performed. A description of the monitor tests is in order.

One of the monitor tests is to detect and log framing and parity errors. This function is performed by first obtaining the time duration of the test from the operator. This step will be performed for each monitor task and the input is verified as digits suitable for a 24 hour clock. The next step is to initialize and activate the SIO channels and the timer. At this point the interrupt driven routines take over, which leaves the monitor software the task of

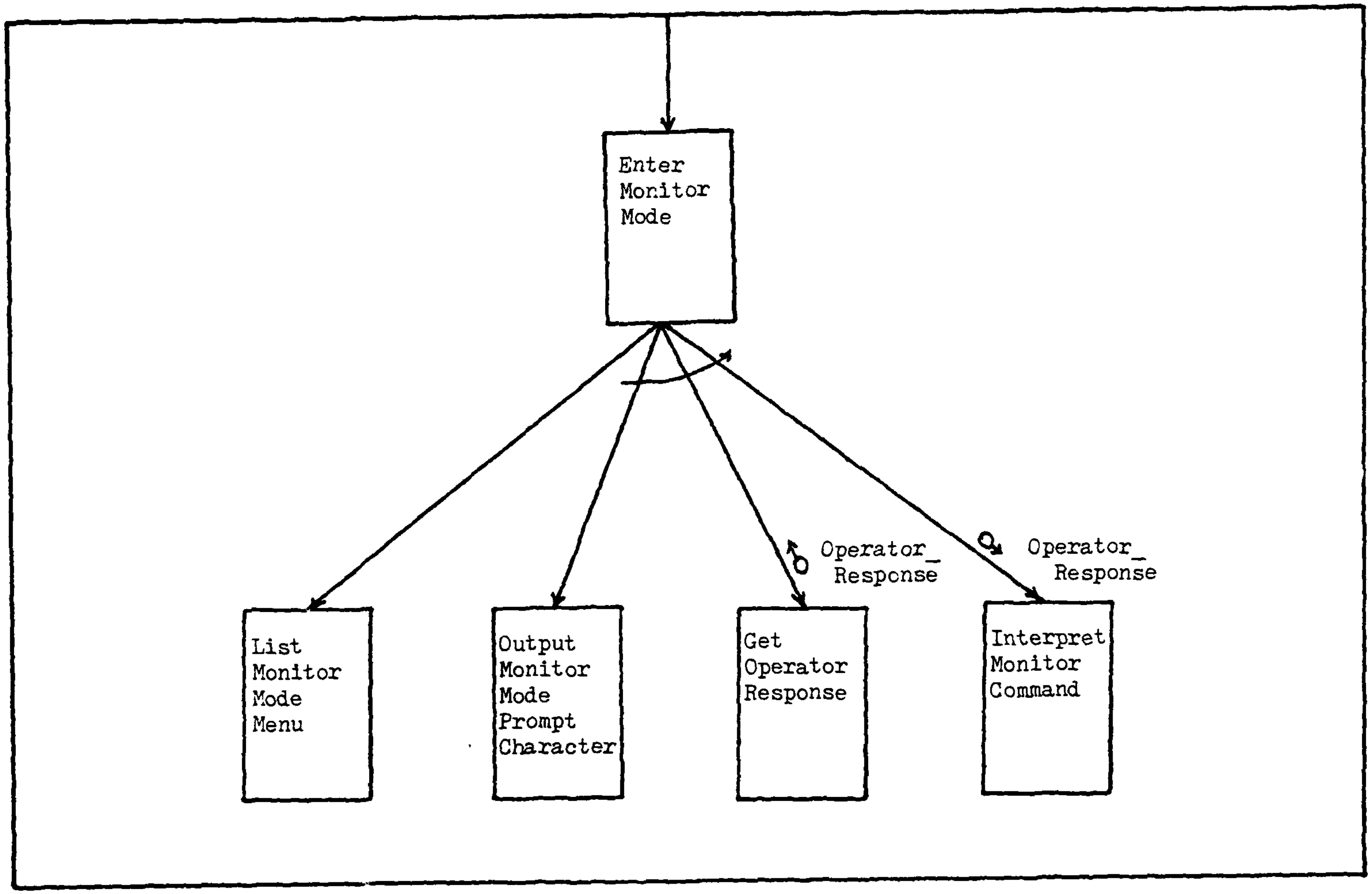


Figure 10. Design of Monitor Mode Sub-Driver (Level 2)

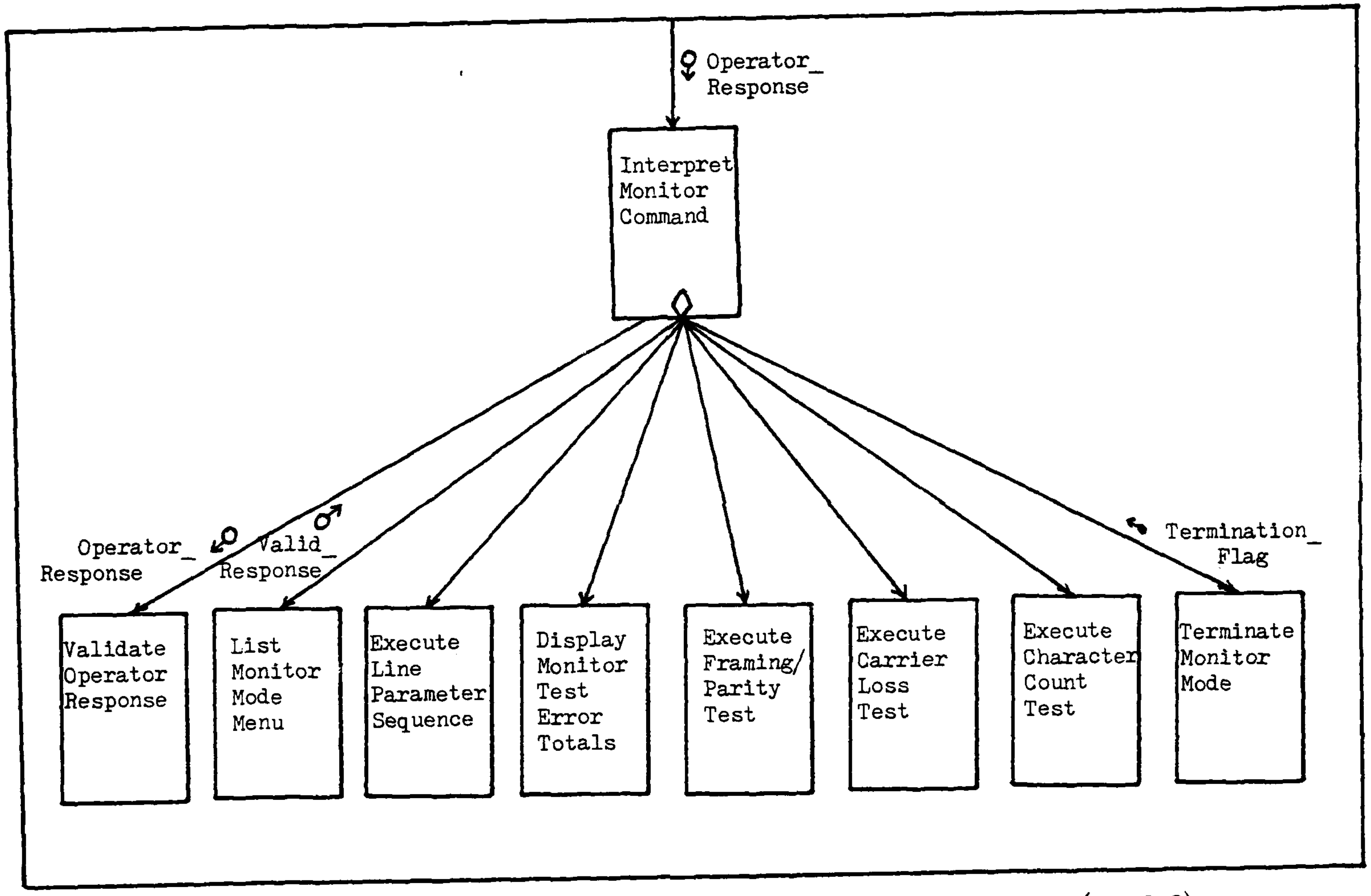


Figure 11. Submodule to Interpret Operator Input at Monitor Mode Level (Level 3)

continually checking the status set by the interrupt routines and main timer runout. Interrupt routines monitor the status of the SIO registers and are triggered by the occurrence of an error. If an error does occur, an interrupt will also occur and the SIO status indicating the type of error will be read and stored for later checking. For example, if an error occurred within the first hour of monitoring, the error will be stored in buffer one of 24 3-byte buffers. The buffers used for each test are separated from the other monitor tests so as to be able to come back to a particular test's results and display them to the operator at will as prescribed by the monitor histogram requirements.

Another monitor task is to detect and log carrier loss errors. The relative time input for measuring the duration of loss is an input obtained from the operator. After initialization and activation of the SIO and the timer, the carrier loss software monitors for a drop of the carrier signal once the signal is initially detected. If the signal comes back up within the loss period set by the operator, the drop is counted as an error. If the signal comes back up after the loss period has expired the drop is not counted as an error but considered a termination of message or communication. A secondary timer of the CTC is used to clock the loss period.

The last monitor task is to detect and log characters received per unit time. This task goes through the same initialization and activation sequence as the previous tasks. All characters received through the SIO are counted and logged within the hour interval the character was received with the duration of the test based on the input from the operator. The interrupt service routines perform the actual character count. At the completion of the time period, these totals are output to not only indicate the end of the test but also to display the results of the test.

In the monitor mode therefore, no data is transmitted, but simply collected to record statuses. Both the receive and transmit lines are monitored. The statuses checked include framing and parity errors and carrier loss errors. One test counts all characters that were received during a specified time interval. The histogram function was not completely designed.

The Simulate Function. The other major task which is performed by the line tester is the simulate mode function. This function allows the operator to transmit data over the communications line; receive the data; and perform comparisons on the data that was sent with the data that was received. The physical interface to support the simulate function will be directly to the communications line. The simulate mode tests range from sending controlled patterns of ones and zeros to messages of various lengths. This function has a sub-driver to control which test is to be performed. The sub-driver allows the execution of simulate mode tests without having to return to the system level for each new test.

The simulate mode function is performed by the module SIMULATE. The upper level design of the SIMULATE module functions is depicted in Figures 12, 13, and 14. The SIMULATE module drives the different tests and calls the specific test to be performed by interpreting the input of the operator. Each simulate mode test is a submodule which is called by module SIMULATE.

The procedure followed in the simulate mode for testing the communications line is to send a full 512-byte buffer of data; receive the data in a second like-size buffer; and then make the comparison between the two buffers. Once the comparison has been completed the process is repeated by transmitting the buffer of data again. The duration of the tests are controlled by the operator. Table I depicts the bit patterns of the simulate mode tests as defined by the functional requirements.

Table I

Simulate Mode Test Bit Patterns

Test	Pattern
Bit Rate	00000000
	11111111
Alternating	01010101
Bit Rate	10101010

One of the simulate mode tests is to perform a bit rate

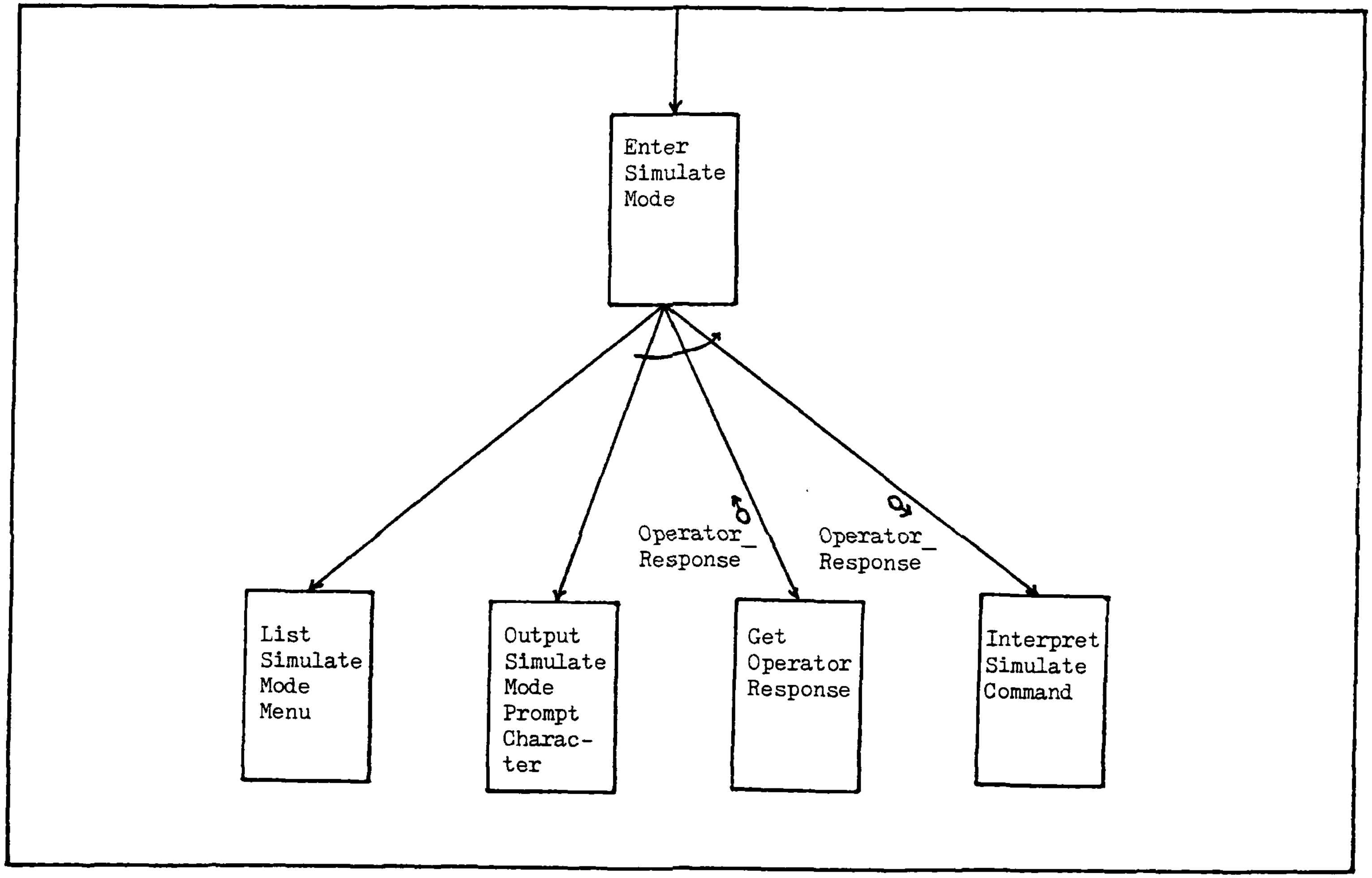


Figure 12. Design of Simulate Mode Sub-Driver (Level 2)

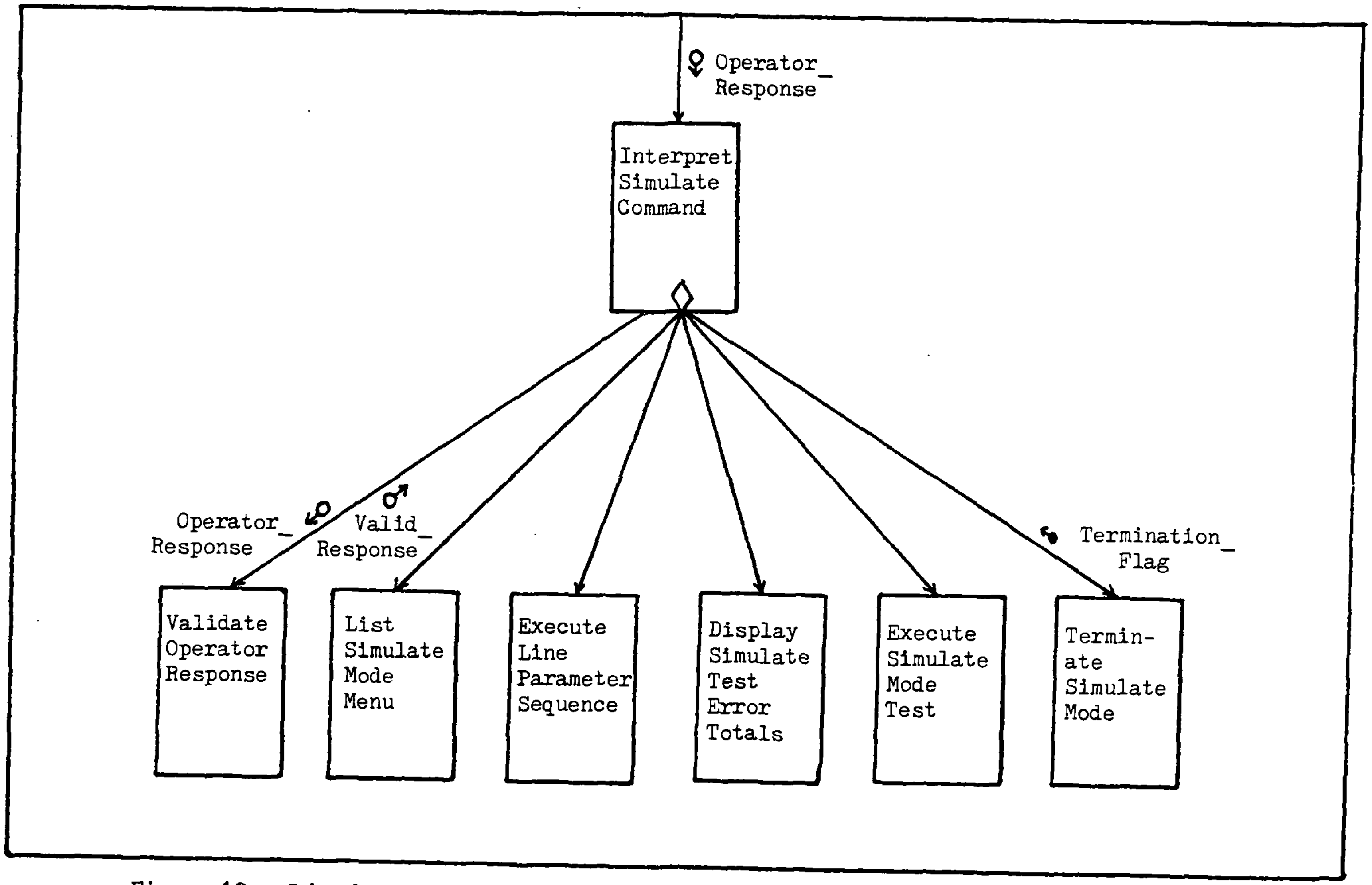


Figure 13. Submodule to Interpret Operator Input at Simulate Mode Level (Level 3)

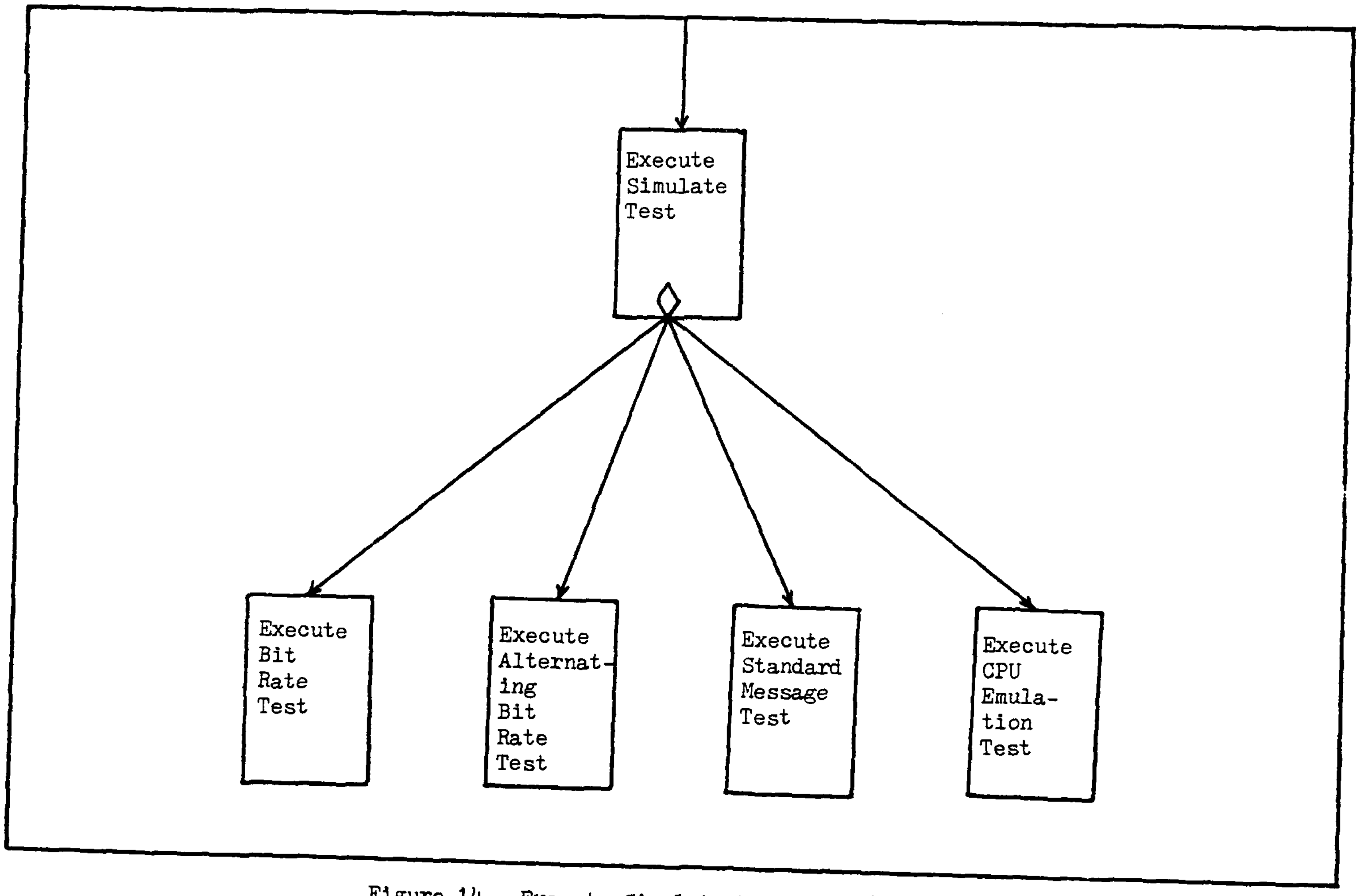

Execute Simulate Test
Execute Bit Rate Test
Execute Alternating Bit Rate Test
Execute Standard Message Test
Execute CPU Emulation Test

Figure 14. Execute Simulate Mode Test (Level 4)

test. As defined by the requirements for this test, a 512-byte buffer consisting of bytes of all ones and bytes of all zeros is sent over the communications line. A loop back device is connected at the other end of the communications line so that the data sent is received again. The received buffer is checked with the transmitted buffer and any errors are logged. An error occurs if the bit pattern sent does not match the bit pattern received. The process is repeated for the relative time specified by the operator.

Another test performed in the simulate mode is the alternating bit rate test. In this test bytes of alternating ones and zeros are sent over the communications line. Buffers of 512 bytes in length are used here also to send and receive. The received buffer is checked with the transmitted buffer and errors are logged. The test is repeated for the time period input by the operator.

A third test performed by the simulate mode is the standard message test. The operator has a choice of inputting the message or sending the message residing in the line tester software which is 'The quick brown fox jumps over the lazy dog.' This message was chosen because it contains every character in the alphabet. The message is sent over the communications line; received and displayed on the user console. The operation may be repeated manually by the operator or set to repeat automatically.

A fourth simulate mode test is to perform CPU emulation or send back to the operator terminal what was input by the operator. This test performs a validation of console input. The character input by the operator is transmitted through the SIO and retrieved via the loop back device for display on the operator console.

In the simulate mode therefore, data is transmitted and received to test the communications line. Bit patterns of ones and zeros are used or messages are sent and displayed to test the line. Table I contains the bit patterns used in the simulate mode tests. An indication of the reliability of the communications line is determined by comparing the received data with the transmitted data or by visual inspection of the data appearing on the operator's console. The histogram function was not completely designed.

The Line Parameter Function. In order to operate in either monitor or simulate mode, various line parameters must be set. These parameters are set by the operator and vary depending on the characteristics of the transmission medium. The line parameter function is needed to establish a dialogue and interface with the operator to receive the inputs for setting the line conditions. This function is achieved by module LINE. The upper level design of the functions of module LINE are depicted in Figure 15. The LINE module is broken out because it is called upon from different levels of the line tester software structure. The line parameters module prompts the operator to input the line parameter settings needed to initialize the SIO interface. The line parameters module is setup to display a

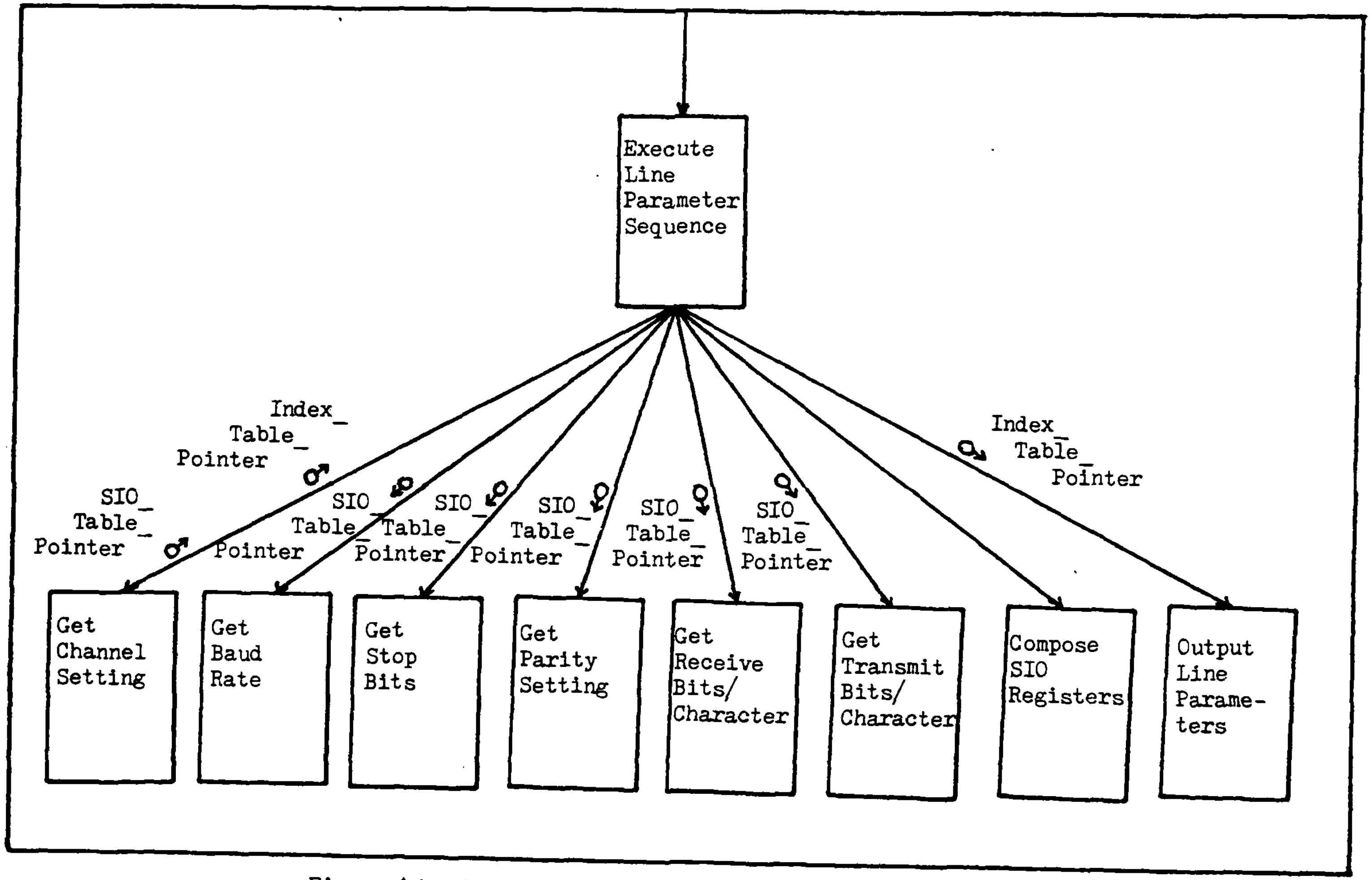


Figure 15. Design of Line Parameter Sub-Driver (Level 2)

list of choices from which the operator may choose. The choices are identified with a letter of the alphabet to make the responses short and easily validated. This form of computer initiated dialogue limited the amount of data required as input from the operator and simplified the software needed to determine the actual input.

The line parameters module executes a sequence of calls to other submodules to obtain the inputs from the operator. This module will initialize either channel of the SIO. The submodules pass a channel pointer to store the input into the proper channel table. Each submodule has error checking to insure that the input is within the range of characters or one of the choices listed. This error checking is important because the response submitted by the operator is modified and used as an index into a table of possible line parameter settings. The table of parameter settings is coded such that when a SIO register specifies several line conditions the parameter settings can be logically 'ORed' together to obtain the desired bit pattern expected in the SIO register. A separate submodule composes the SIO register values and stores them into the SIO table. The SIO table is later used by initialization routines in the monitor and simulate mode to activate the SIO channels.

Pointers and index tables were used in the LINE module to facilitate the retrieval of current line parameter settings. Excessive string comparisons would have been necessary otherwise.

The Utilities Function. There are many routines which both the simulate and monitor modes utilize to perform tasks. The function here then is to provide support to the monitor and simulate mode tasks. As was mentioned earlier, the SIO is the center of activity for both modes of operation; therefore, the majority of the tasks performed by the utilities function concern the SIO. The SIO will operate in an interrupt driven environment; therefore, all interrupt driven routines, which service the SIO and CTC interrupts, have been collected in the utilities module for use by both the monitor and simulate modes. Appendix E, which refers to SIO programming, provides guidance on the complexities of SIO initialization and interrupt structure. In addition, the utilities module contains SIO initialization drivers and timer drivers both upon which monitor mode and simulate mode call.

The utilities module is therefore a collection of service routines and not truly a functionally independent module. It is best then to describe its submodules completely. The design of the submodules of the UTILITIES module may be found in Appendix C.

The SIO initialization drivers take data stored in the SIO tables and write the values to the SIO registers in a set sequence. The SIO becomes activated once all registers have been written into. Contained in this module also are the routines to reset the SIO table values to a known state before execution of the line parameters sequence which alters these values.

The timer drivers activate the counter timer chip (CTC) and set them to interrupt every second which allows for the test duration to be set in increments of one second. Interrupts once a second can be easily counted by the software and provide an accurate relative time. A routine monitors the interrupts to ascertain relative timing. Since the interrupt of the timer is approximate the clocking of the tests will not be all that accurate but within reasonable ranges. Deviations of one minute or two over an hour period may be experienced.

There exist eight other interrupt driven routines that service SIO interrupts. These routines are pointed to by an interrupt table just as the CTC routines. The interrupt table is located in RAM to allow for changes in the addresses. Flexibility is achieved this way by being able to point to any desired interrupt service routine by simply writing the address of the routine in the interrupt table. Modifying of the interrupt table is done in the line tester software to better service simulate mode functions. The simulate mode works with buffers whereas the monitor mode discards the characters.

The eight other interrupt driven routines service the SIO interrupts for valid character receive, invalid character receive, external/status, and transmit character interrupts for each channel (Ref 4:33-40). There exists an interrupt table which contains the addresses of each interrupt routine. The SIO modifies the interrupt vector address according to the type of interrupt which may have occurred. Therefore, for example, the receive character routine will only be called if a valid character was received by the SIO. The receive character interrupt routines accept the character from the SIO and discard it. For the monitor mode of operation the receive character interrupt routines do not use buffers. Different receive character interrupt routines are used in the simulate mode. For the simulate mode of operation the buffer pointer is allowed to increment through the entire buffer since the data is to be kept for later checking.

The invalid receive character interrupt routines store the error status before discarding the character. The error status is obtained from registers within the SIO and is reset after it is read. Both monitor mode and simulate mode use the status while the character in this routine is discarded in the monitor mode and kept in the simulate mode.

The external/status interrupt routine services interrupts caused by changes in modem control signals. The status is read from the SIO and stored for checking by the monitor mode of operation. The status is reset to allow current values to appear again in the status register. The monitor mode carrier loss test uses this routine to determine when a carrier signal has come back up after dropping.

The transmit interrupt routine transmits the character whenever the transmit buffer becomes empty. The character to be transmitted is found in a buffer which was loaded prior to enabling the transmit operation.

The use of interrupt driven routines allows for greater freedom of the CPU to be performing other tasks. Currently the line tester software is dedicated to performing one task at a time and the freedom is not really needed. The efficiency of the interrupt driven routines has been utilized to allow for expansion of the line tester functions.

Summary

The material discussed in this chapter covered the various design aspects of the line tester software. An analysis of the software environment was first performed to establish a starting point in the software design. Following the environment analysis, several techniques of software design were discussed. The techniques presented were used to aid in satisfying the general requirements of software modularity and maintainability. Finally the design of each of the functions to be performed by the line tester was discussed. The software modules performing the functions along with module interfaces are summarized in Figure 16. The functions were designed by modules to satisfy the requirements and to aid in managing software implementation.

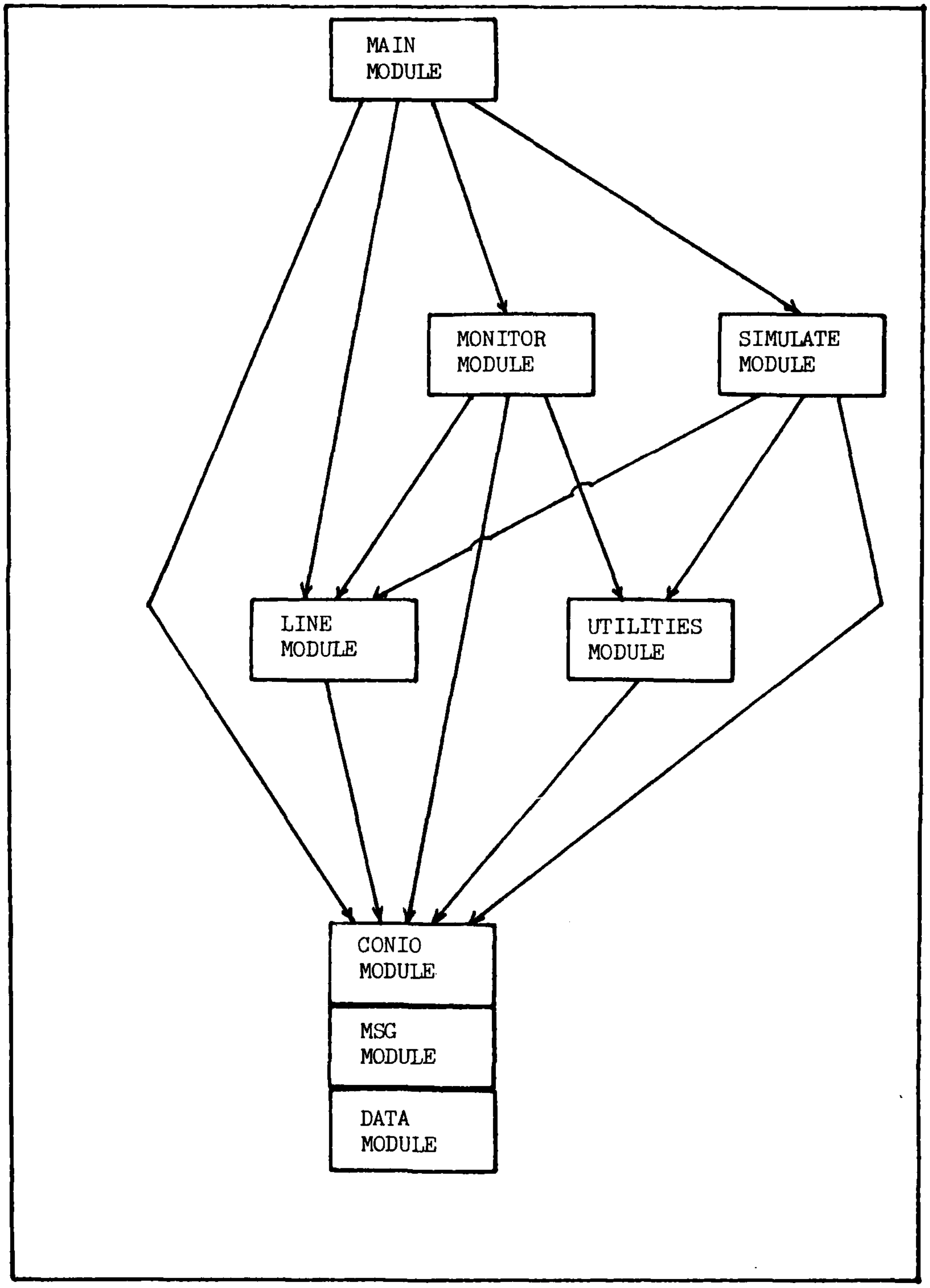


Figure 16. Overview of Module Interfaces

IV. System Implementation and Testing

Introduction

The software design of the previous chapter was implemented on the Mostek Z-80 system. The modularity of the software permitted many implementation options. The software could be grouped so as to accomodate the hardware limitations of the PROMs.

The design emphasizes the man-machine interface because of its importance in retrieving the proper input parameters for the tests. The performance of the line tester software depends upon how well the operator understands the inputs needed for program execution. Extensive error checking is performed to insure that the inputs are within the specified values.

Other design points are that the outputs to the operator are numerous to provide an indication of what is going on during the execution of the test, and that maximum flexibility is provided to allow for future enhancements. Each test is designed to be as independent as possible of the other tests. This independence allows for use of existing software drivers when adding new modules.

Implementation and testing was performed systematically. Each level was implemented and tested prior to the next level. The drivers were tested first to establish a foundation to test all other line tester functions in a top-down fashion. Testing revealed the need to change the design of a few modules which were again tested. The restructuring, when needed, was easy because of the modular design. Design changes usually involved breaking up larger modules into smaller ones or moving of modules from one level to another or within a level.

Implementation

The software was developed on the Zilog Z-80 development system located in the Digital Engineering Laboratory of AFIT. Testing and debugging was accomplished within the Mostek Z-80 system provided, as mentioned earlier, by the Weapons Laboratory. The software was transferred from the Z-80 development system to the Mostek by utilizing an existing software transfer program which was designed foı transfers of files from the Z-80 system to the Universal Network Interface Device (UNID) (Ref 2).

Even though the entire line tester software package was tested in RAM, booting and initialization of variables was performed as if the line tester software existed in PROM. Testing of the booting capabilities was performed this way since programming the line tester into PROM would not occur until the Weapons Laboratory obtained the software. The Zilog Z-80 development system did not have a PROM programming capability for the type PROMs used on the Mostek system.

A memory map is provided in Table II and indicates the contents of memory on the Mostek system. The software will

Table IJ

Mostek Z-80 Memory Configuration

Starting and Ending Adresses in Hex	Contents
FF00-FFFF	Stack Pointer (256X8 Static RAM)
F000-FEFF	Not Used
E800-EFFF	Not Used
E000-E7FF	User Program (2K PROM)
D800-DFFF	User Program (2K PROM)
D000-D7FF	User Program (2K PROM)
C800-CFFF	User Program (2K PROM)
C000-C7FF	User Program (2K PROM)
B000-BFFF	Not Used
A000-AFFF	Not Used
9000-9FFF	Not Used
8000-8FFF	Not Used
7800-7FFF	User Program Variables (2K RAM)
7000-77FF	Available (2K RAM)
6000-6FFF	Available (4K RAM)
5000-5FFF	Available (4K RAM)
4000-4FFF	Available (4K RAM)
3000-3FFF	Available (4K RAM)
2000-2FFF	Available (4K RAM)
1000-1FFF	Available (4K RAM)
0000-0FFF	Available (4K RAM)

begin at location E000 (Hex) to take advantage of a hardware strapping of a restart to this location.

As mentioned earlier, all variables will reside in RAM. Also located in RAM is the interrupt table used by the SIO and CTC. Locating this table in RAM allowed for additional flexibility in handling the interrupts of the SIO. Many special purpose routines could be called upon by substituting addresses in the table. The peculiarities of the simulate mode of operation were handled in this manner.

Relative timing was achieved by programming the CTC to interrupt after each second. This allowed for an easy way to count the elapsed time using software routines. These CTC interrupt service routines also reside in the interrupt table with the SIO routines.

Implementation of the interface software with the operator resulted in the command list shown in Table III and detailed in the users manual of Appendix D. Single character phrases as opposed to multiple character phrases were chosen to delineate housekeeping commands from actual test commands which are multiple character. Each level of line tester software prompts the operator with the commands upon entering that level and then only if the operator issues a command to see the list of commands again will they be displayed.

The command designations such as the letter 'M' for monitor mode, have been selected to be closely associated with the function they perform. In addition, a prompting

Table III

Quick Reference of Commands

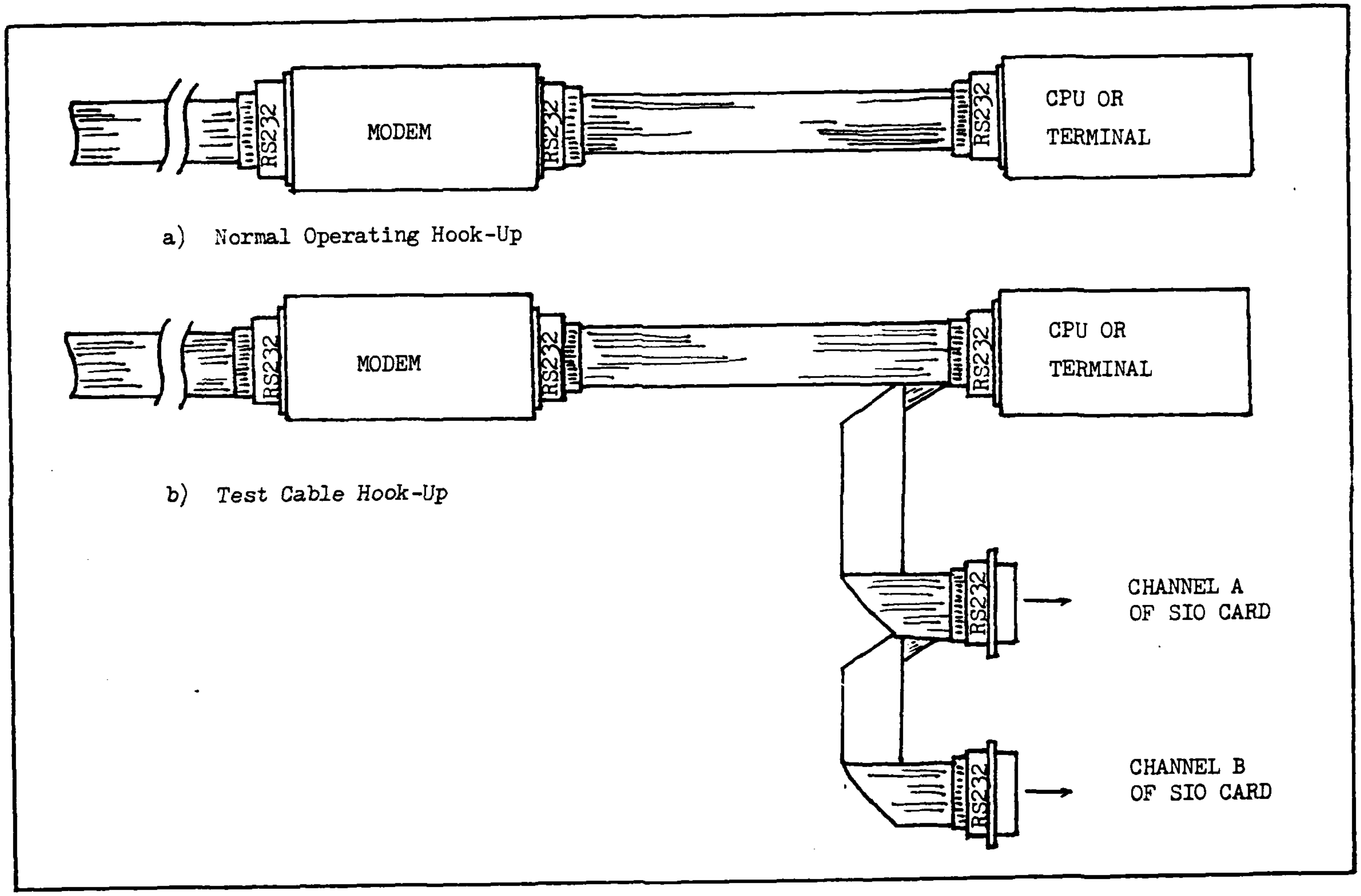
Command Abbreviation	Name
M	Monitor (Prompt = %)
S	Simulate (Prompt = >)
D	Display
P	Parameter (Prompt = *)
Q	Quit
H	Histogram
LOGFP	Framing and Parity Test
LOGCL	Carrier Loss Test
LOGCC	Character Count Test
LOGBR	Bit Rate Test
LOGAB	Alternating Bit Rate Test
LOGSM	Standard Message Test
LOGCE	CPU Emulation Test

character such as a dollar sign ($) for the system level and a percent sign (%) for the monitor mode, indicate to the operator the particular level being executed.

The operator's console will be connected to the Mostek through the use of the UART, and the SIO will be connected to the communications line to enable testing of that communications line. Monitoring will be performed by using the cable setup depicted in Figure 17. Each SIO channel will be provided information, and as indicated in Figure 18 one will receive the 'receive' information of the line while the other will receive the 'transmit' information of the line. SIO interrupt routines will handle the reception and transmission of data along these lines.

Testing

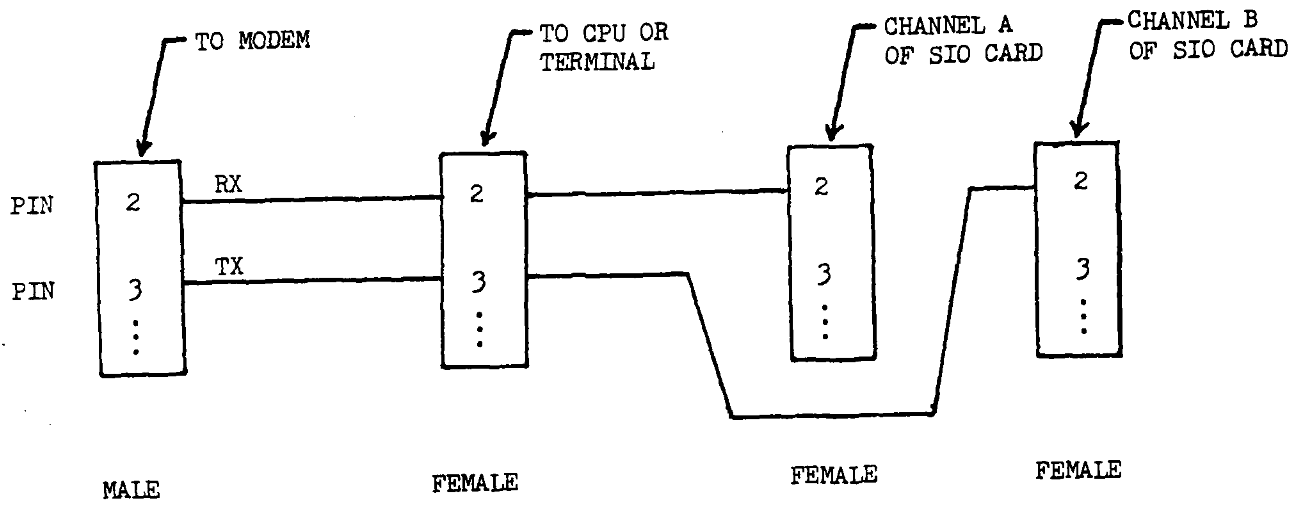
Testing was performed to locate possible errors in the design of the software. Complete testing of software is impossible but several test strategies were used to introduce a degree of confidence in the performance of the implemented design. During the testing phase several principles were followed. One principle was to define the expected output before performing the test. In other words, the output was determined first and then the test was performed to see if indeed the software provided the same correct output. Another principle followed was that the results were thoroughly examined. There is a tendancy to look for only the right answer amongst the test results.



a) Normal Operating Hook-Up

b) *Test Cable Hook-Up*

Figure 17. Physical Cable Connections for Monitor Mode



* NOTE: PINS 2 AND 3 ARE THE ONLY TWO PINS THAT HAVE BEEN MODIFIED ON THE RS232 TEST CABLE

Figure 18. Pin Configuration For Cable Setup

Finally, people unfamiliar with the program were asked to use the line tester. This tested the ability of the line tester software to handle erroneous inputs.

The initial strategy chosen to test the software was black box testing (Ref 16:8). The intent of black box testing is finding circumstances in which the program does not behave according to the specifications. This strategy is unconcerned with the internal structure of the program.

One form of black box testing used was equivalence partitioning (Ref 16:44). In this class of testing, a small subset of all possible inputs is selected. This subset is chosen so that the maximum classes of inputs are tested. Classes used in testing were inputs of valid responses and invalid responses. In addition, when a range of values was required as an input, the test values used were numbers within the range and numbers outside the range of values.

Another form of black box testing was boundary value testing (Ref 16:50). In this class, the inputs would be at the extremes of a range of values and just beyond the range. An attempt was also made to achieve tests where the outputs of the software were also made to be at the extremes of buffer limitations or time limits.

A second strategy chosen was white box testing (Ref 16:9). In this strategy the internal structure of the program was tested. Test data was used to attempt to test all decision paths within the software's internal structure.

Summary

The implementation of the line tester software was accomplished through systematic procedures. No long, troublesome manipulations are required to get the software up and running. (Simply turn the machine on.) The man-machine interface software makes the software an easily managed product.

Testing of the line tester software revealed design problems which were resolved by re-coding or restructuring the module in which the design error occurred. Both black box and white box testing strategies were used. These strategies provided a high confidence level in the line tester software.

V. Recommendations and Conclusions

Introduction

The intent of this thesis investigation was to develop software for a particular purpose. The purpose was for asynchronous data communications line testing. Since the goal was achieved, not much can be said about the impact of the project except to mention items which were involved with the testing of communications lines. Also, the application of line testing to new environments, such as embedding the line testing capability in network systems, can be explored.

Recommendations

This thesis investigation covered asynchronous data communications. An obvious next step would be the development of a synchronous communications line tester. Another item of concern in the field of data communications which goes one step beyond testing and error detection is error correction which is quite possible with the speeds of current microprocessors. There is a need for the ability to handle burts of noise, clicks, distortion, and high background noise on communications lines such that the message being sent is received correctly. These are some of the causes of data errors on communications lines. Microprocessor-based systems could assist in recognizing the cause and correcting the error.

The Z-80 assembly language proved to be an efficient language for programming the line tester functions. The block handling instructions of the Z-80 language would be equally suitable for use on the error correction problem previously discussed. The powerful instruction set of the Z-80 language effectively supports communications processing.

Conclusions

The microprocessor is a very capable tool for dealing with line testing. Microprocessors are being used in ever increasing numbers, and their use has in part been necessitated by the similar increases in the capabilities of data processing systems. The microprocessor-based line tester developed in this thesis investigation was an effective tool with which to satisfy the requirements set by the Weapons Laboratory. As specified by the Weapons Laboratory, two modes of operation were implemented for use in asynchronous data communications line testing. In addition, man-machine interface software was developed to allow the operator to interactively control the testing functions. Design of the line tester software was structured to allow for ease of maintenance. The tests desired by the Weapons Laboratory and implemented via the line tester software include monitoring and simulation of the communications line. Monitor mode tests log errors involving carrier signal loss and framing and parity errors. In addition, a test will count all characters transmitted across the communications line. Simulate mode tests transmit data over the line and compare what was sent with what was received.

The microprocessor can perform well in a limited applications environment such as the data line testing utilization by the Weapons Laboratory. One can easily see that the microprocessor, being an extremely flexible device, can be tailored to fulfill a vast number of similar communications problems and tasks.

The top down approach for the design of the line tester software was ideal in that the approach allowed for the development of simple, efficient modules. In addition, the transaction oriented design strategy idealy modeled the functions which the line tester was to perform.

Much attention was given the man-machine interface because of its importance in communicating with the operator. Error checking routines developed were critical in insuring that the correct parameter was set for a testing operation.

Computer initiated dialogue aided in reducing the error checking required on operator input. The inputs of the operator were somewhat controlled by this dialogue which served to limit the possible variations of the input and thereby limit the amount of error checking. In addition, menus were provided to help the operator in selecting the desired response.

The transmission and reception of data was interrupt driven to provide additional efficiency. Efficiency resulted from not tying up the CPU which would have been constantly checking the status flags of the CTC and SIO had not interrupts been implemented. Any future enhancements can take advantage of the efficiency provided by the interrupt structure to perform other tasks.

In conclusion, the microprocessor-based line tester is a flexible and versatile tool for the testing of data communication lines. The utilization of microprocessors discussed in this thesis provides a stepping stone to other tasks in the data communications environment.

BIBLIOGRAPHY

1. AFWL-TR-77-68. Air Force Microprocessor Technology Applications. Final Report submitted to AF Weapons Laboratory by R&D Associates. Marina del Rey, CA: R&D Associates, 1977.

2. Baker, Lee R. Prototype and Software Development for Universal Network Interface Device. MS Thesis. Wright-Patterson AFB, OH: School of Engineering, Air Force Institute of Technology, December 1980.

3. Burton, H. O. and D. D. Sullivan. "Errors and Error Control," Computer Communications, edited by Paul E. Green, Jr. and Robert W. Lucky. New York, NY: IEEE Press, 1975.

4. Kane, Jerry. An Introduction To Microcomputers: Volume 3 Some Real Support Devices. Berkeley, CA: Adam Osborne and Associates, Inc., September 1978.

5. Knuth, Donald E. The Art of Computer Programming: Fundamental Algorithms, Volume 1. Reading, MA: Addison-Wesley Publishing Company, 1973.

6. Kretzmer, E. R. "The New Look in Data Communications," Computer Communications, edited by Paul E. Green, Jr. and Robert W. Lucky. New York, NY: IEEE Press, 1975.

7. Leventhal, Lance, A. Z-80 Assembly Language Programming. Berkeley, CA: Adam Osborne and Associates, Inc., 1979.

8. Martin, James. Systems Analysis For Data Transmission. Englewood Cliffs, CA: Prentice-Hall, Inc., 1972.

9. Meadow, Charles T. Man-Machine Communication. New York, NY: John Wiley and Sons, Inc., 1970.

10. MK79624. Dynamic RAM Module MDX-DRAM. Operations

Manual for Mostek RAM. Carrolton, TX: Mostek Corporation, 1978.

11. MK79604. EPROM/UART Module MDX-EPROM/UART. Operations Manual for Mostek UART. Carrollton, TX: Mostek Corporation, 1980.

12. MK79742. MDX-SIO Rev A Serial Input/Output Module. Operations Manual for Mostek SIO. Carrollton, TX: Mostek Corporation, 1979.

13. MK79707. Mostek 1979 Microcomputer Data Book. Product catalog. Carrollton, TX: Mostek Corporation, 1979.

14. MK79608. Serial Input/Output Module MDX-SIO. Operations manual for Mostek SIO. Carrollton, TX: Mostek Corporation, 1978.

15. MK79612. Z-80 Central Processing Module MDX-CPU1. Operations manual for Mostek Z-80. Carrollton, TX: Mostek Corporation, 1978.

16. Myers, Glenford J. The Art of Software Testing. New York, NY: John Wiley and Sons, 1979.

17. Rubey, Ray. Lecture materials distributed in EE6.93, Software Engineering. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, 1981.

18. Sellers, Frederick F., Jr., et al. Error Detecting Logic For Digital Computers. New York, NY: McGraw-Hill Book Company, 1968.

19. Softech, Inc. An Introduction to SADT--Structured Analysis and Design Technique. Technical Report 9022-78R. Softech, Inc. Waltham, MA. November, 1976.

20. Weinberg, Victor. Structured Analysis. New York, NY: Yourdon Press, 1979.

21. Yourdon, Edward and Larry L. Constantine. Structured Design: Fundamentals of a Discipline of Computer

Program and Systems Design. New York, NY: Yourdon Press, 1978.

22. Zilog. Z-80 Assembly Language Programming Manual. Cupertino, CA: Zilog Inc., 1977.

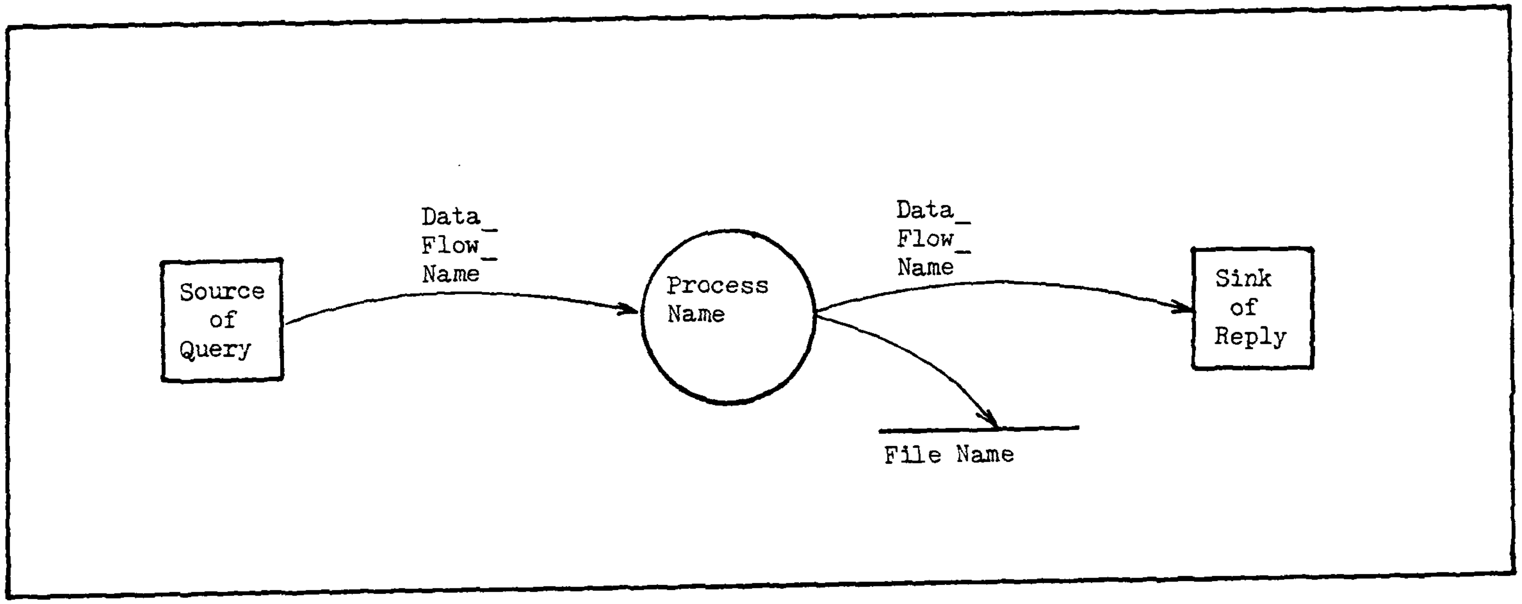
Appendix A

Data Flow Diagrams

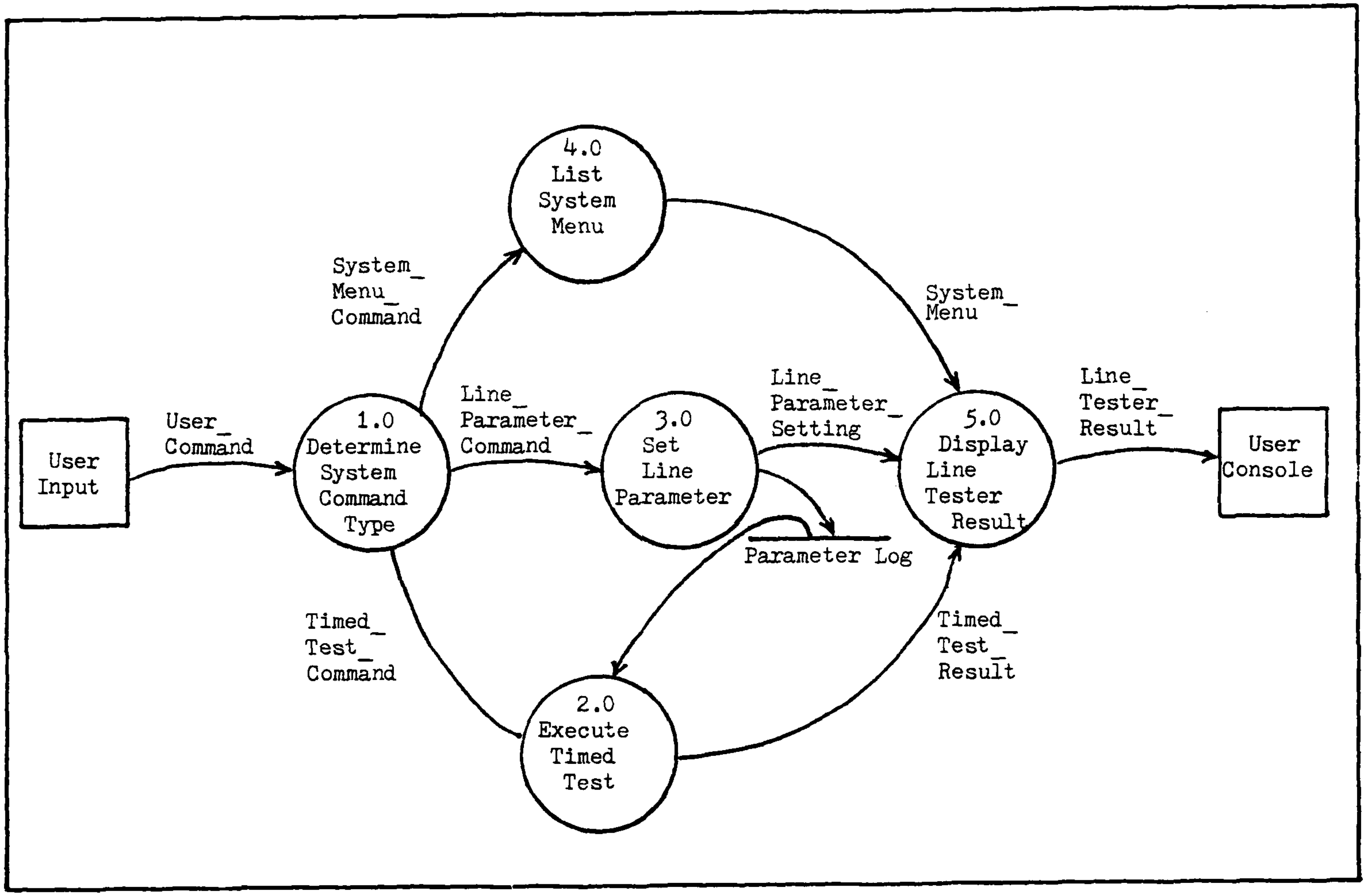
This appendix contains the complete set of data flow diagrams which were used to define the requirements in the requirements definition chapter.

The system diagram represents the top level or complete set of functions which the line tester is to perform. The other data flow diagrams are representations of parts of the system diagram. Contained within this appendix are the DFDs are listed below with indentations indicating the heirarchy of requirements.

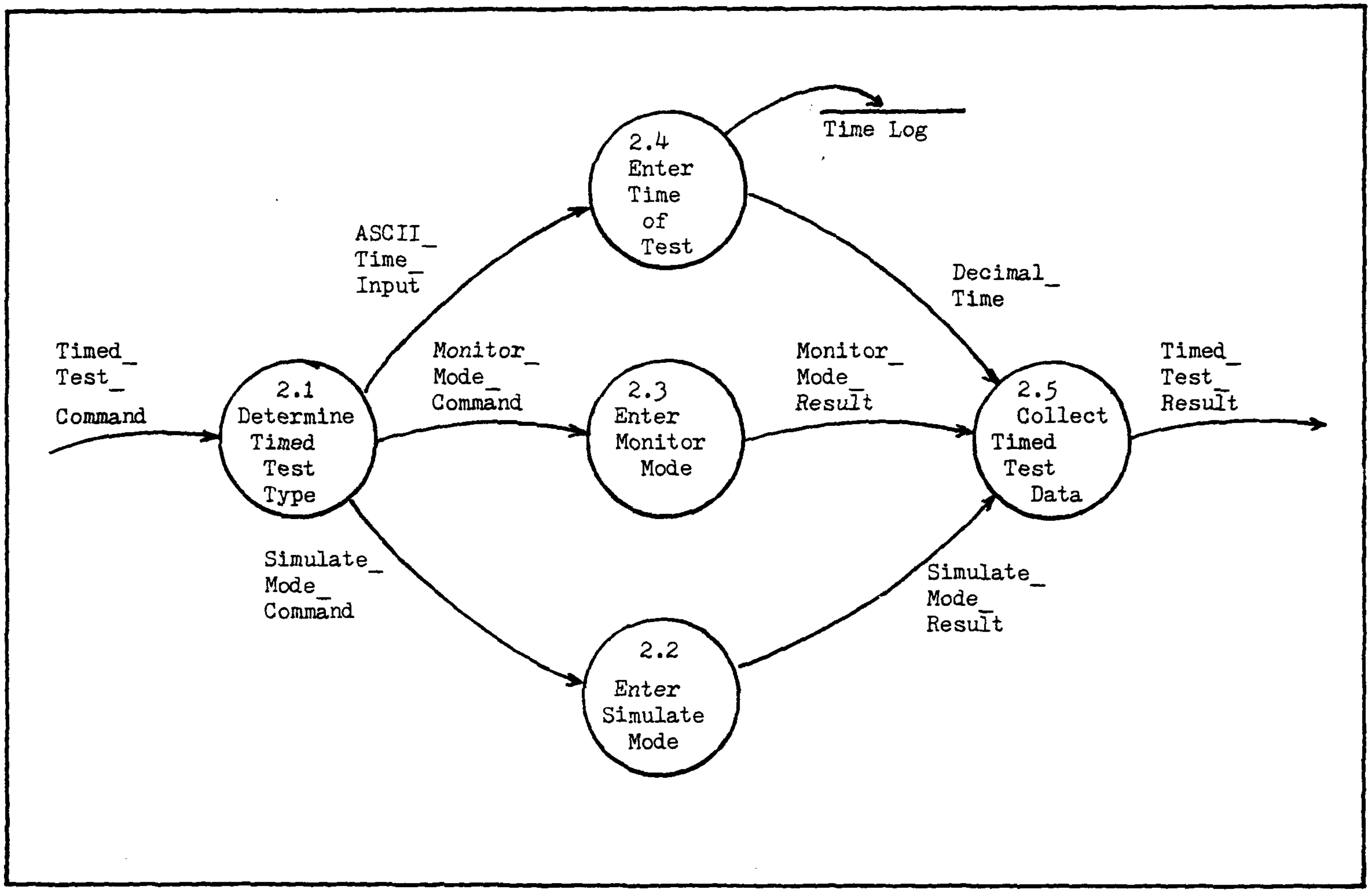
- System Diagram
  - Execute Timed Test
    - Enter Monitor Mode
      - Perform Monitor Mode Test
    - Enter Simulate Mode
      - Perform Simulate Mode Test



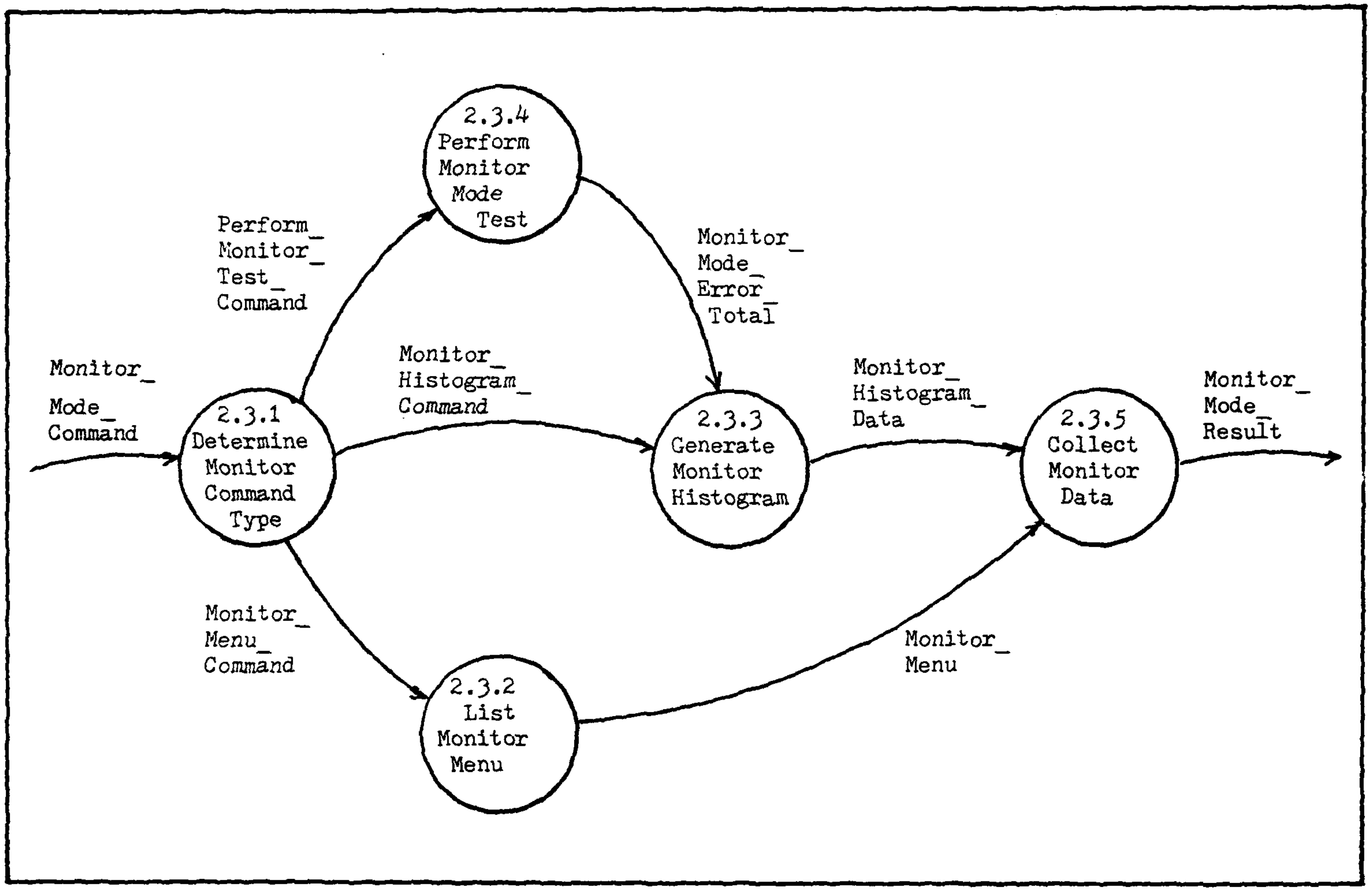
Data Flow Diagram (DFD) Components



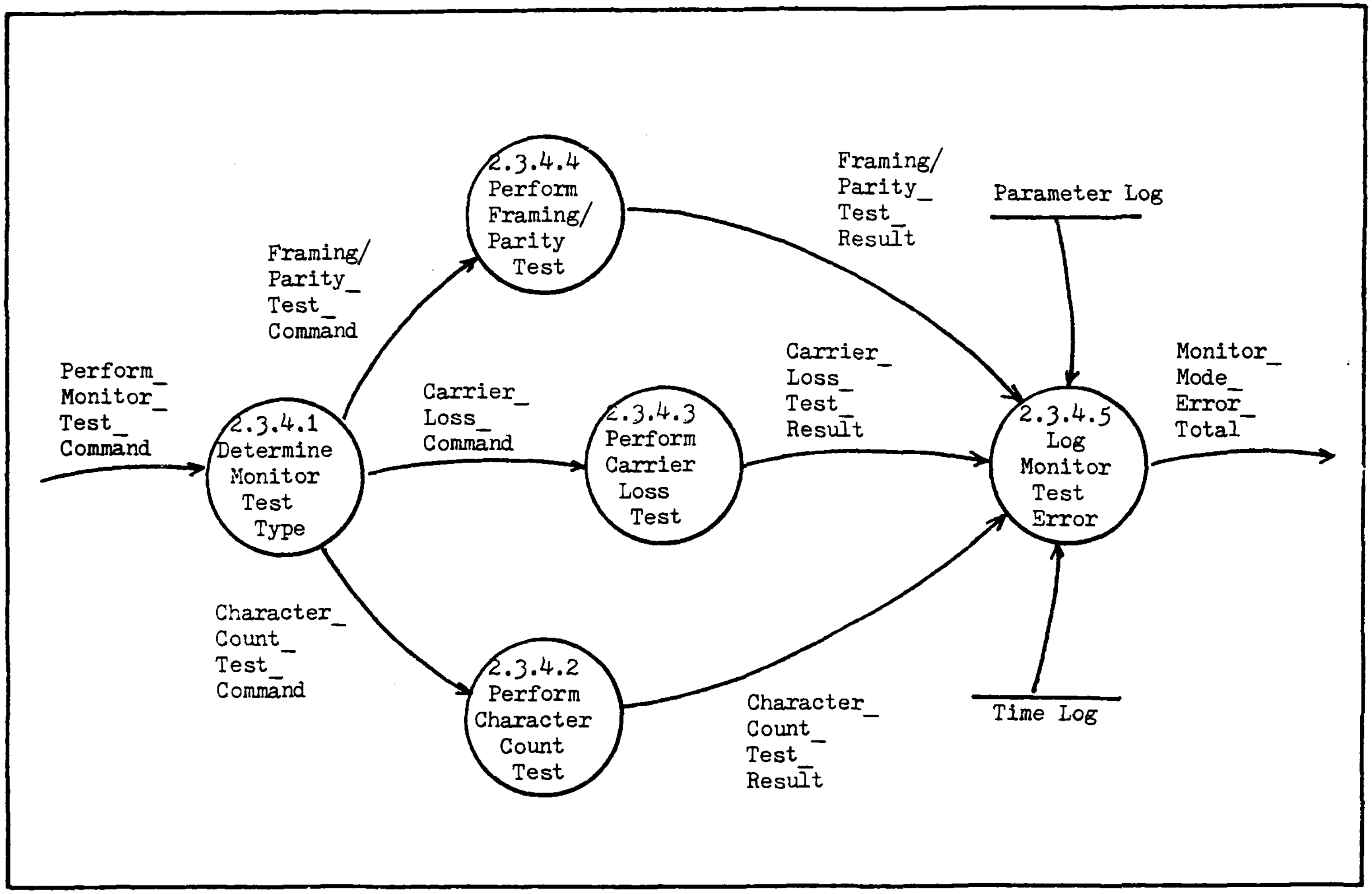
System Diagram--Perform Data Communications Line Test (Level 0)



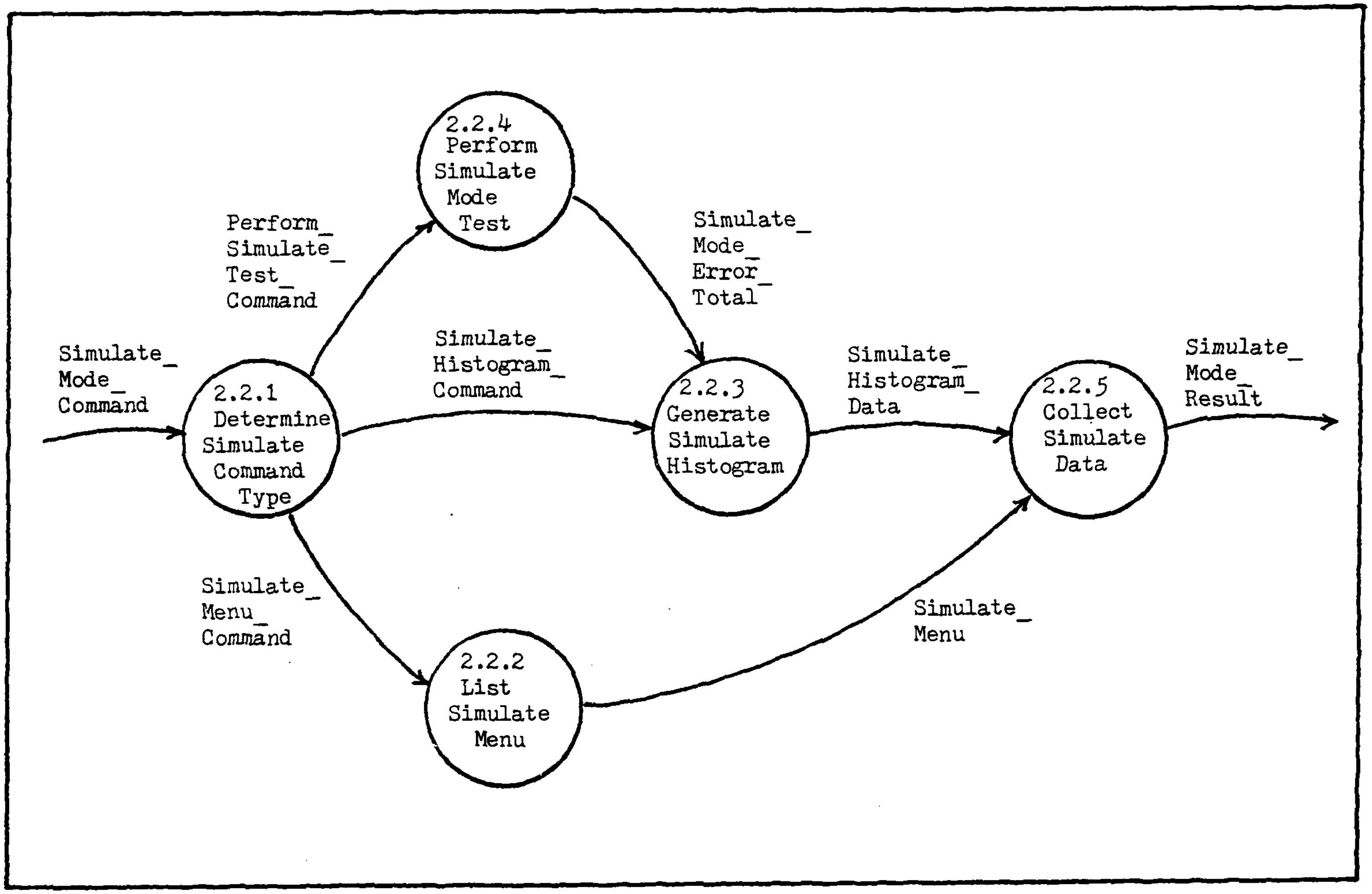
Execute Timed Test (2.0) DFD (Level 1)



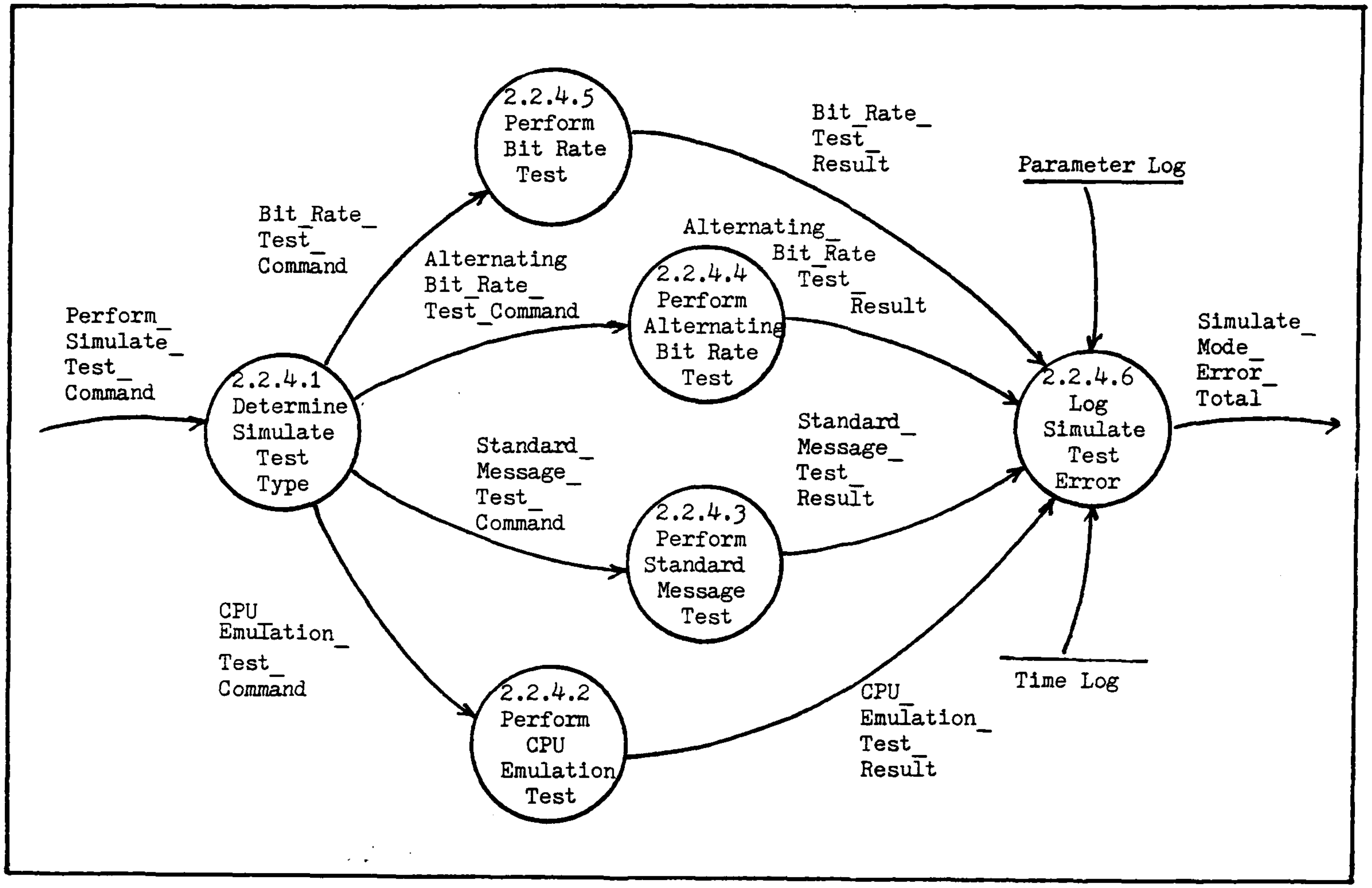
Enter Monitor Mode (2.3) DFD (Level 2)



Perform Monitor Mode Test (2.3.4) DFD (Level 3)



Enter Simulate Mode (2.2) DFD (Level 2)



Perform Simulate Mode Test (2.2.4) DFD (Level 3)

Appendix B

Mostek Z-80 Hardware Components

This appendix lists those hardware components on which the software was implemented and tested. The hardware described in this appendix was provided by the Weapons Laboratory. Refer to the Mostek 1979 Microcomputer Data Book which is published by the Mostek Corporation of Carrollton Texas for further details. A block diagram of the hardware is provided at the end of this appendix.

Component

Z-80 Central Processor Module (MDX-CPU1-4)

Part No. MK77850-4

Features

1. Z-80 CPU
2. 4K x 8 EPROM sockets (two 2716's)
3. 256 x 8 bytes Static RAM
4. Memory decoding jumper selectable
5. Four counter/timer channels
6. Restart to 0000H or E000H (Strapping option)
7. 4 MHZ clock
8. +5 volts only
9. Standard Z-80 bus compatible

Component

Dynamic RAM Module (MDX-DRAM32-4)

Part No. MK77752-4

Features

1. 32K bytes of dynamic RAM
2. Selectable addressing on 4K boundaries
3. 4 MHZ version
4. Standard bus compatible

Component

EPROM/UART Module (MDX-EPROM/UART-4)

Par+ No. MK77753-4

Features

1. 10K x 8 EPROM/ROM sockets (2716's)
2. Serial I/O channel
3. RS232 and 20 mA interface
4. Reader Step control for teletype
5. Baud rate generator 110-19200 Baud
6. 4 MHZ version
7. Standard bus compatible

Component

Serial Input/Output Module (MDX-SIO-4)

Part No. MK77651-4

Features

1. Two independent full duplex channels
2. Independent programmable Baud rate clocks
3. Asynchronous data rates--110 to 19.2K bits/sec
4. Receive data register quadruply buffered
5. Transmitter data register double buffered
6. Asnychronous operation
7. Binary synchronous operation

8. HDLC or IBM SDLC operation
9. Both CRC-16 and CRC-CCITT hardware implemented
10. Modem control
11. Operates as DTE or DCE
12. Serial input and output as either RS232 or 20 mA current loop
13. Current loop optically isolated
14. Current loop selectable for either active or pasive mode
15. Address programmable
16. 4 MHZ version
17. Compatible with standard Z-80 bus

Component

Card cage (MD-CC8)

Part No. MK77954

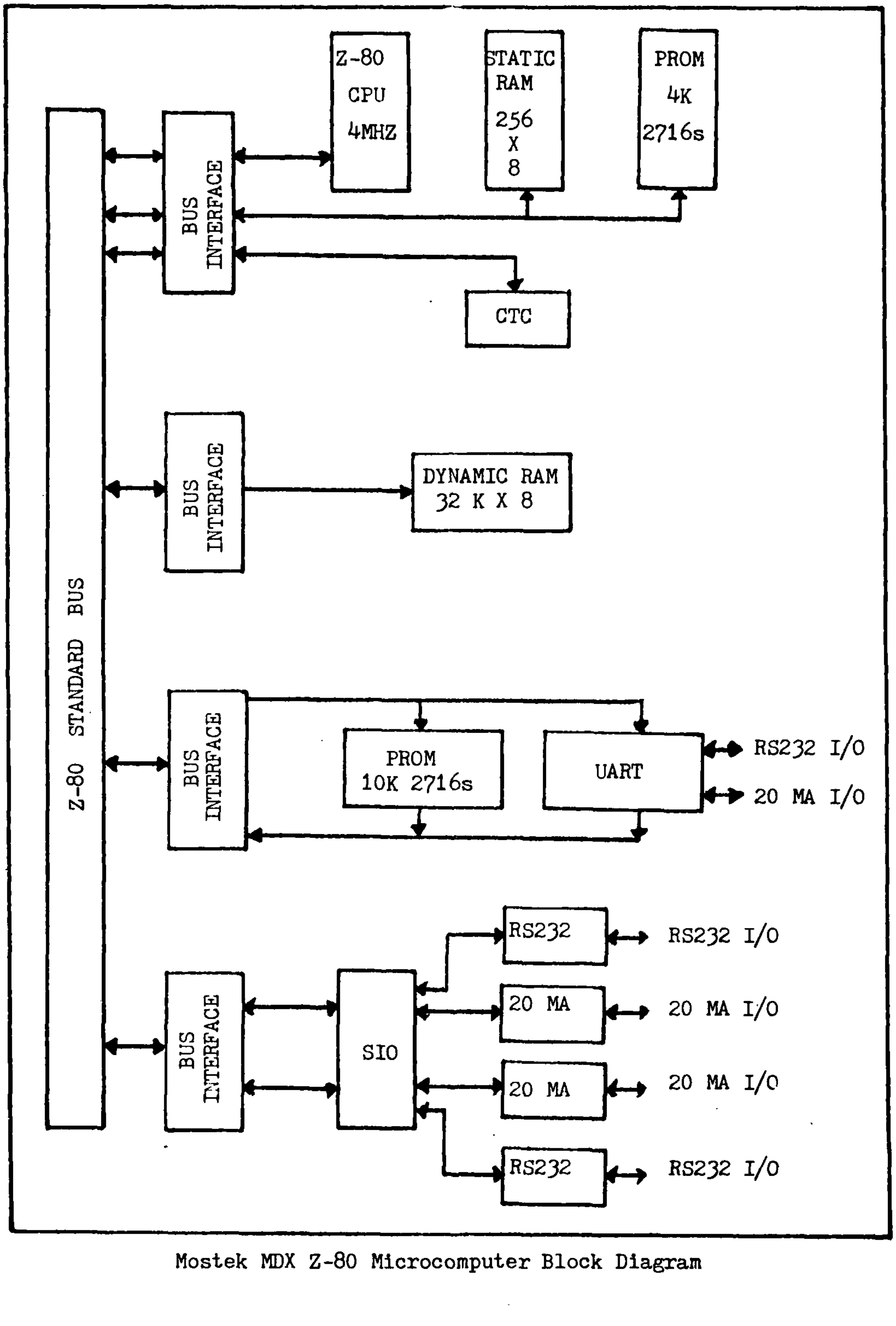
Features

1. 8 slot
2. Standard bus mother board

AD-A115 560 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DEVELOPMENT OF A MICROPROCESSOR-RASED ASYNCHRONOUS DATA COMMUNI--ETC(U)
DEC 81 D L HARTMANN
UNCLASSIFIED AFIT/GCS/EE/81D-11 NL

2 of 2
AD
A115560

END
DATE
FILMED
7 82
DTIC



Mostek MDX Z-80 Microcomputer Block Diagram

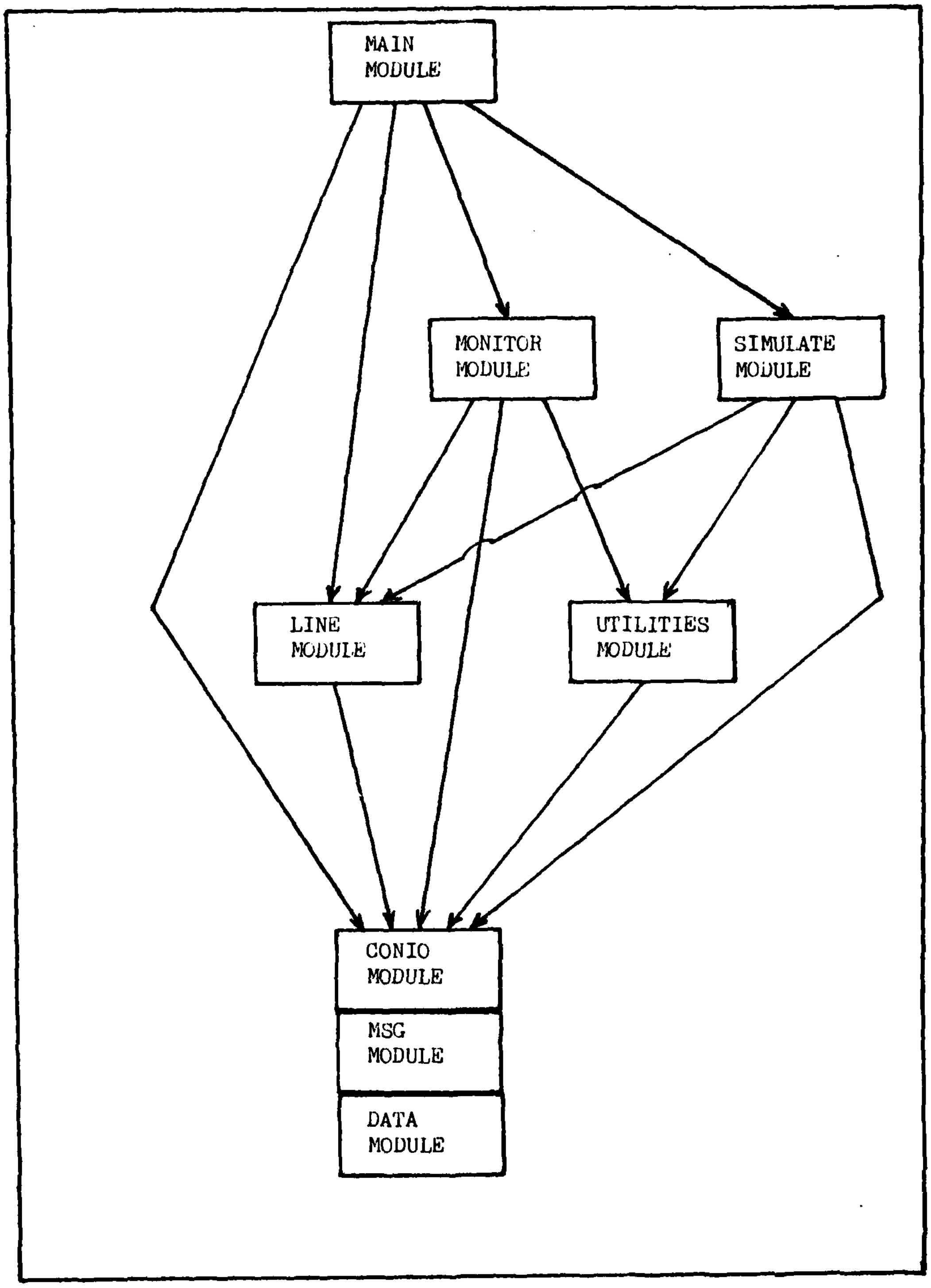
Appendix C

Structure Charts

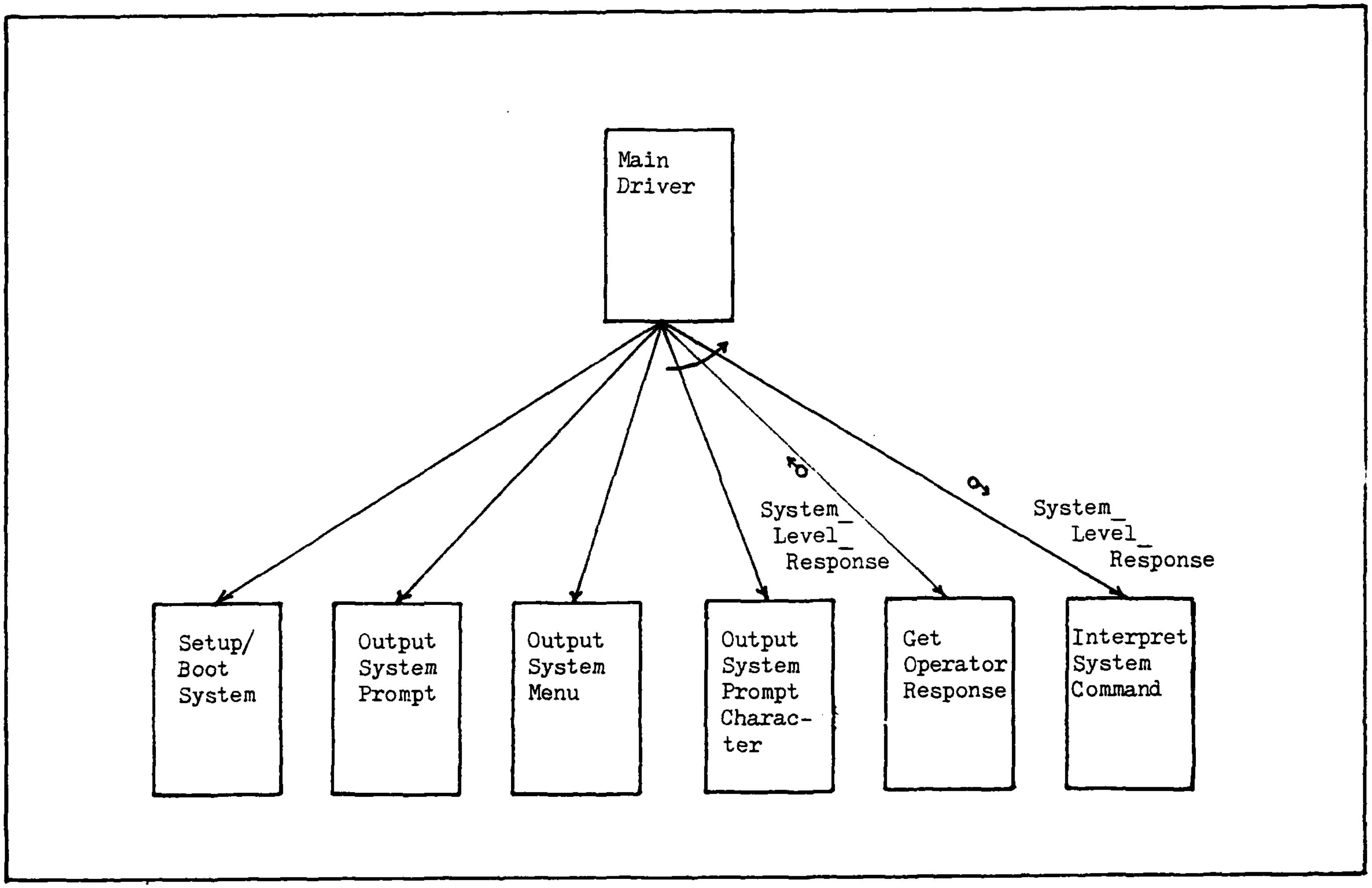
This appendix contains the complete set of structure charts which depict the design of the line tester software. The interrupt driven modules will appear as unconnected structures at the end of this appendix.

The components of the structure charts include lines, boxes, and annotated arrows. The boxes represent processes or modules which perform some function as indicated by phrases within the box. The lines joining the boxes represent relationships between processes. The arrows depict what kind of relationship exists between the processes. An arrow with a closed circle indicates a control flag being passed between the processes while an arrow with an open circle indicates the passing of data between processes. Lines with no arrows represent process relationships with no data being passed.

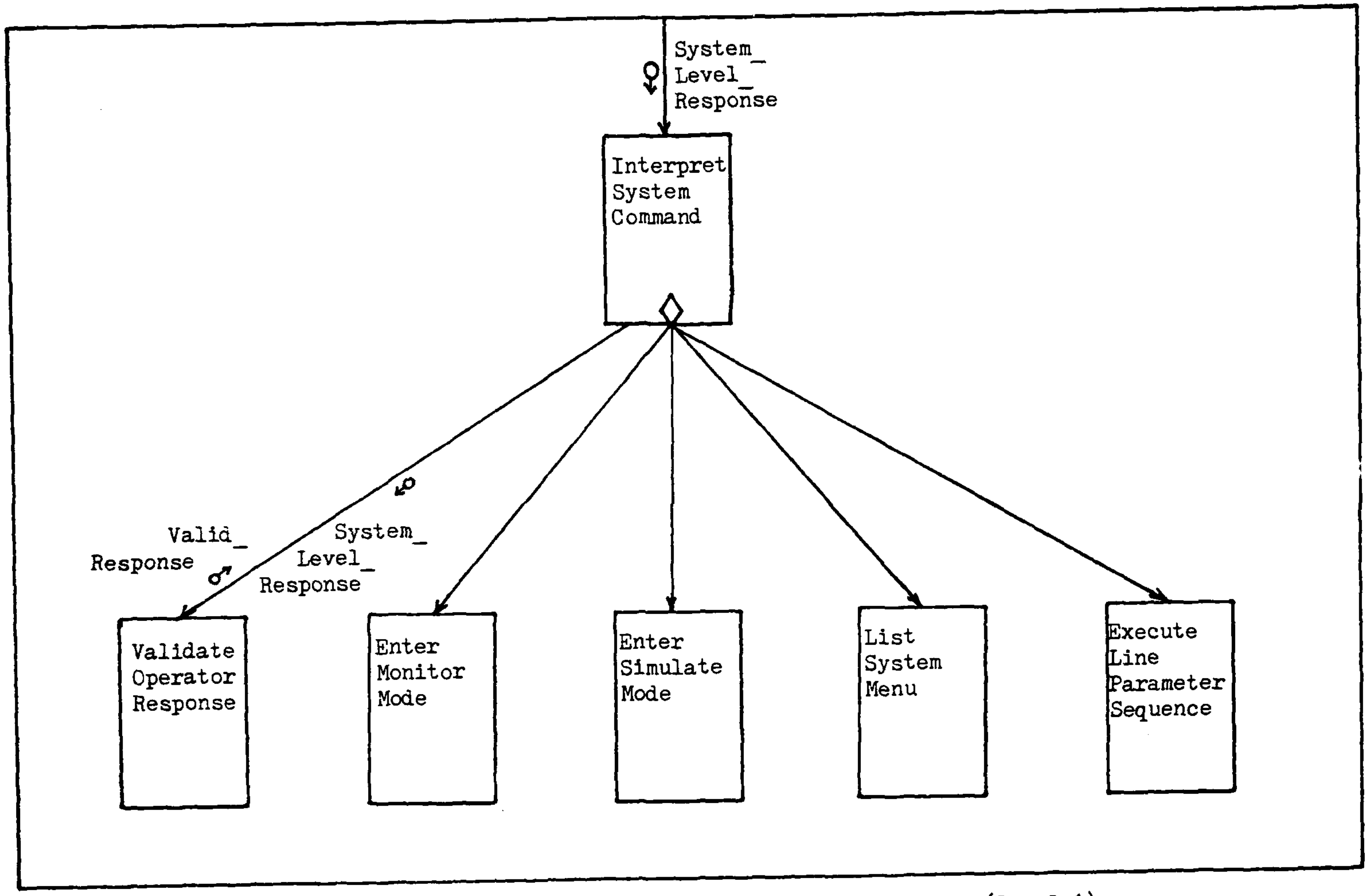
Contained within this appendix is the design of the line tester software as depicted by structure charts. The modules are in the following sequence: MAIN, CONIO, LINE, UTILITIES, MONITOR, SIMULATE. Level numbers are given to indicate how far down the structure, beginning with module MAIN, the chart is located. All modules can be followed back to module MAIN except for the interrupt routines which as stated earlier are unconnected structures.



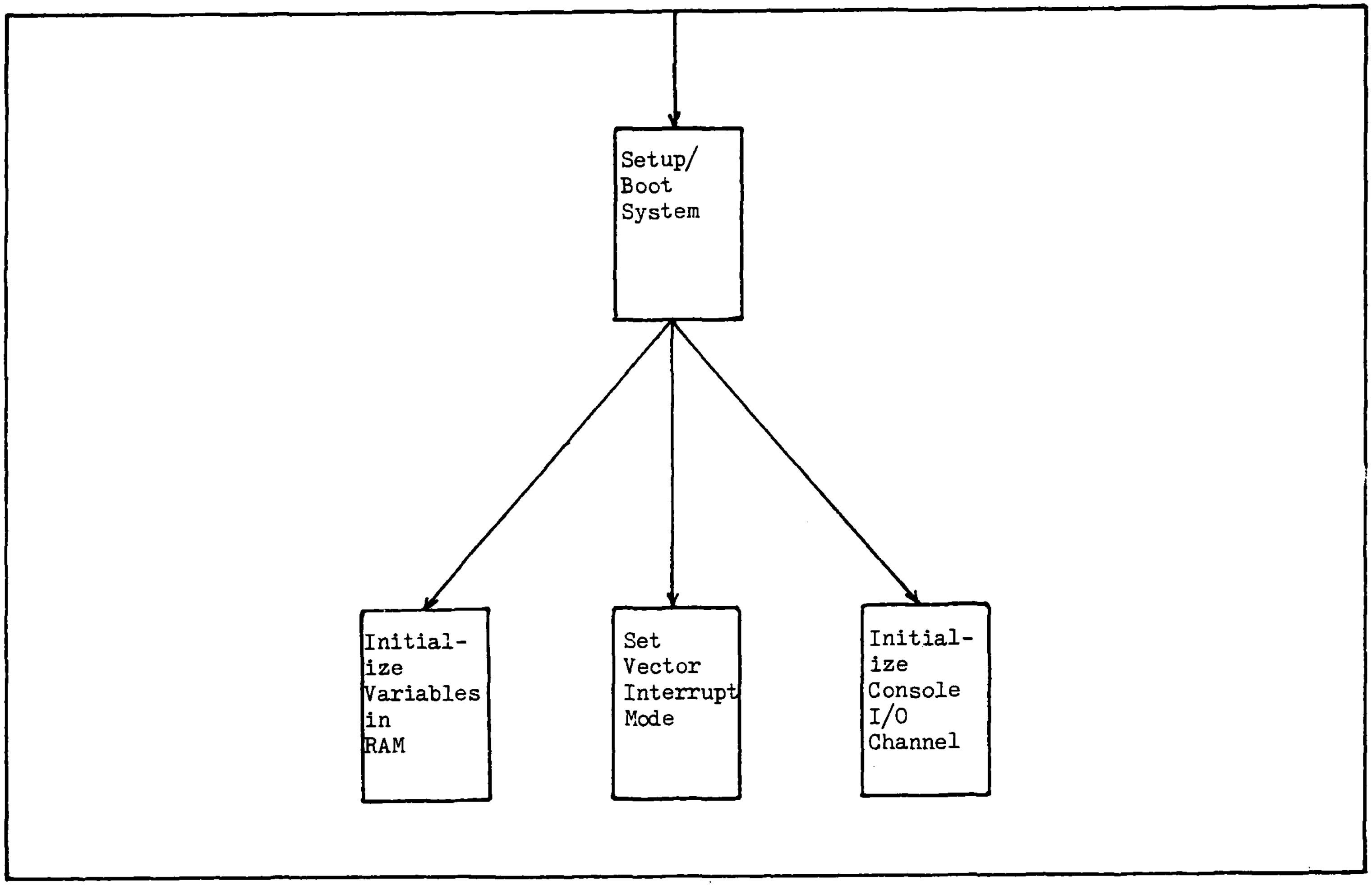
Overview of Module Interfaces



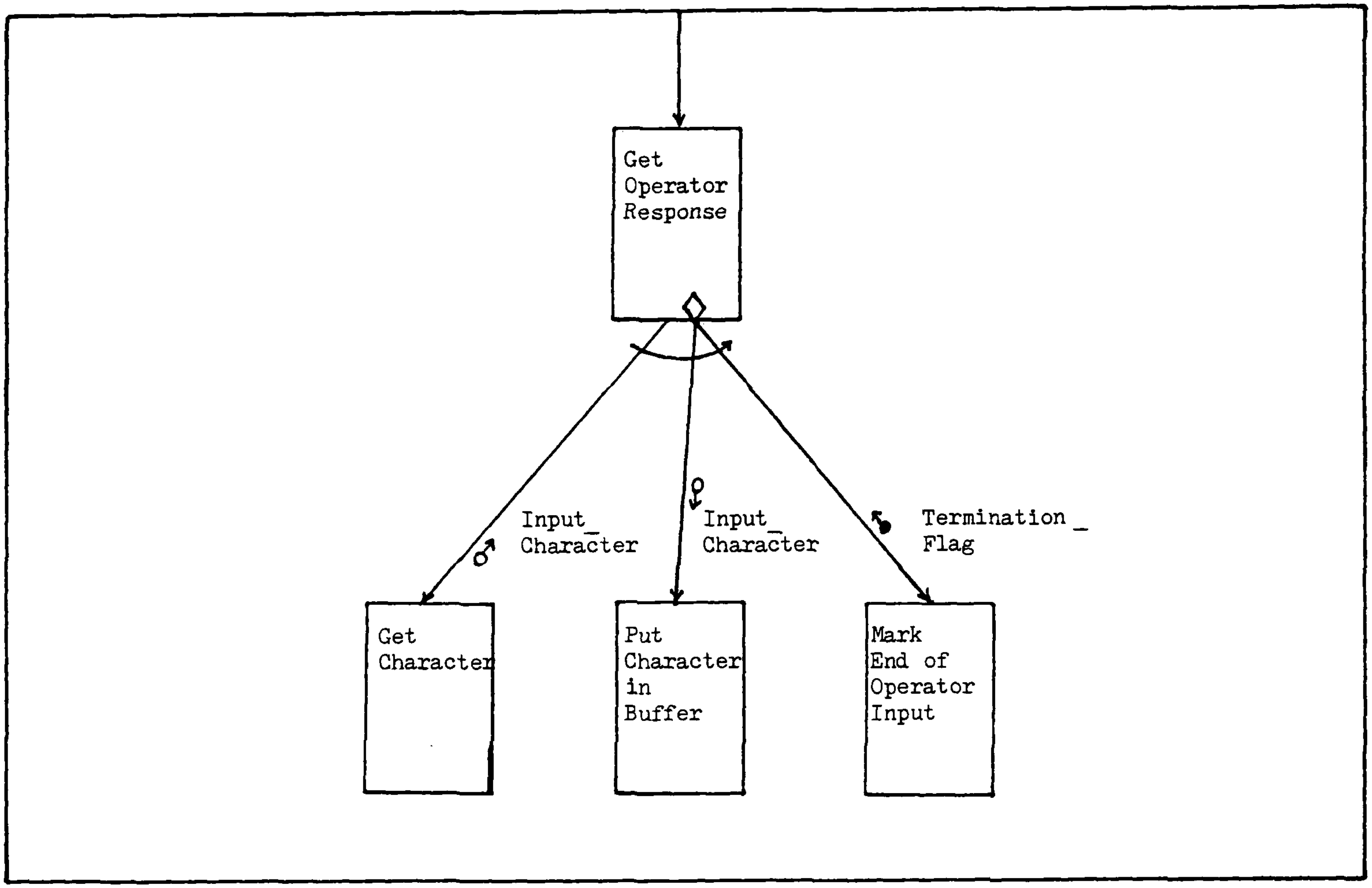
MAIN--Design of Main Driver for Line Tester (Level 0)



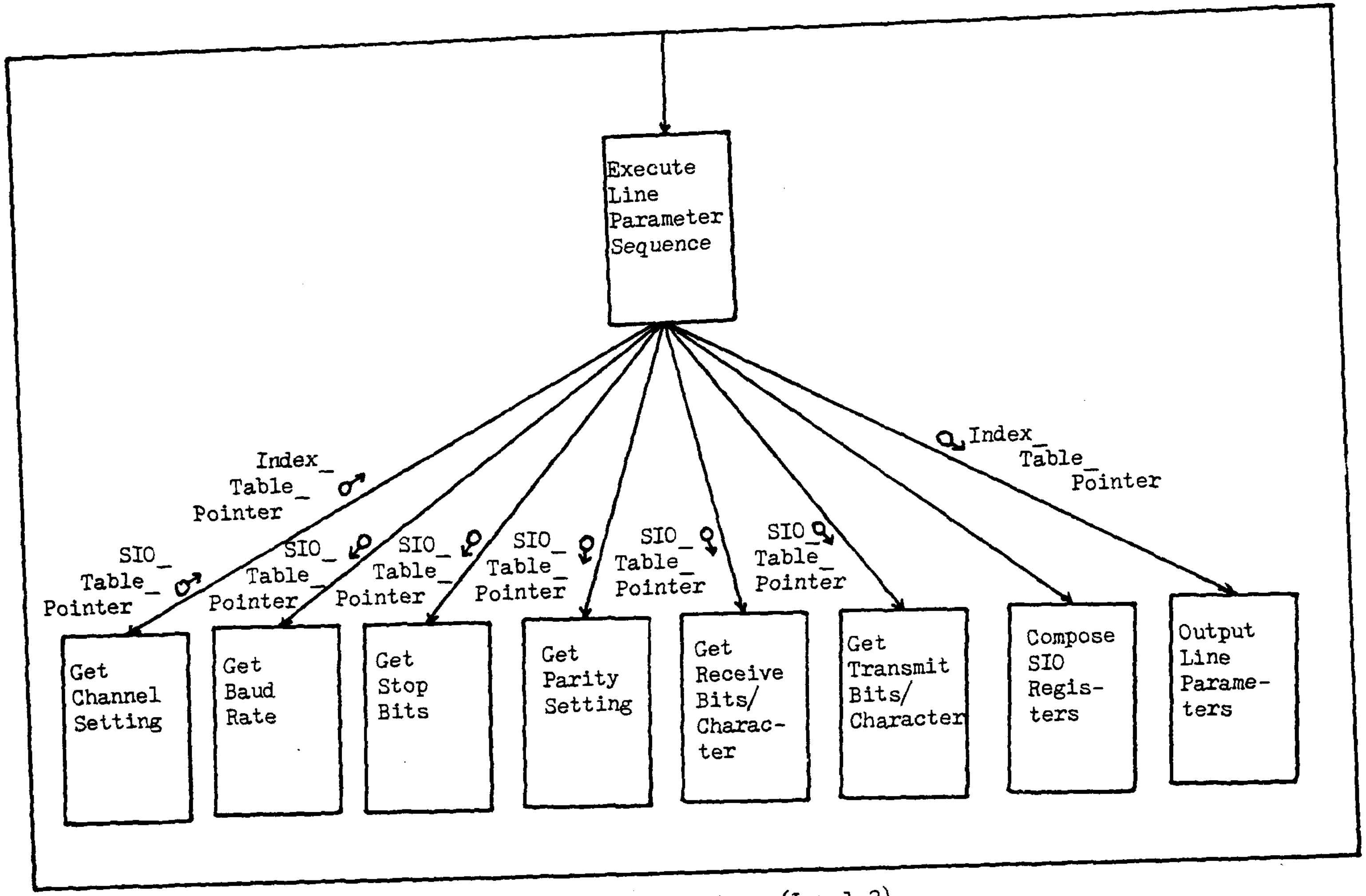
MAIN--Submodule to Interpret Operator Input at System Level (Level 1)



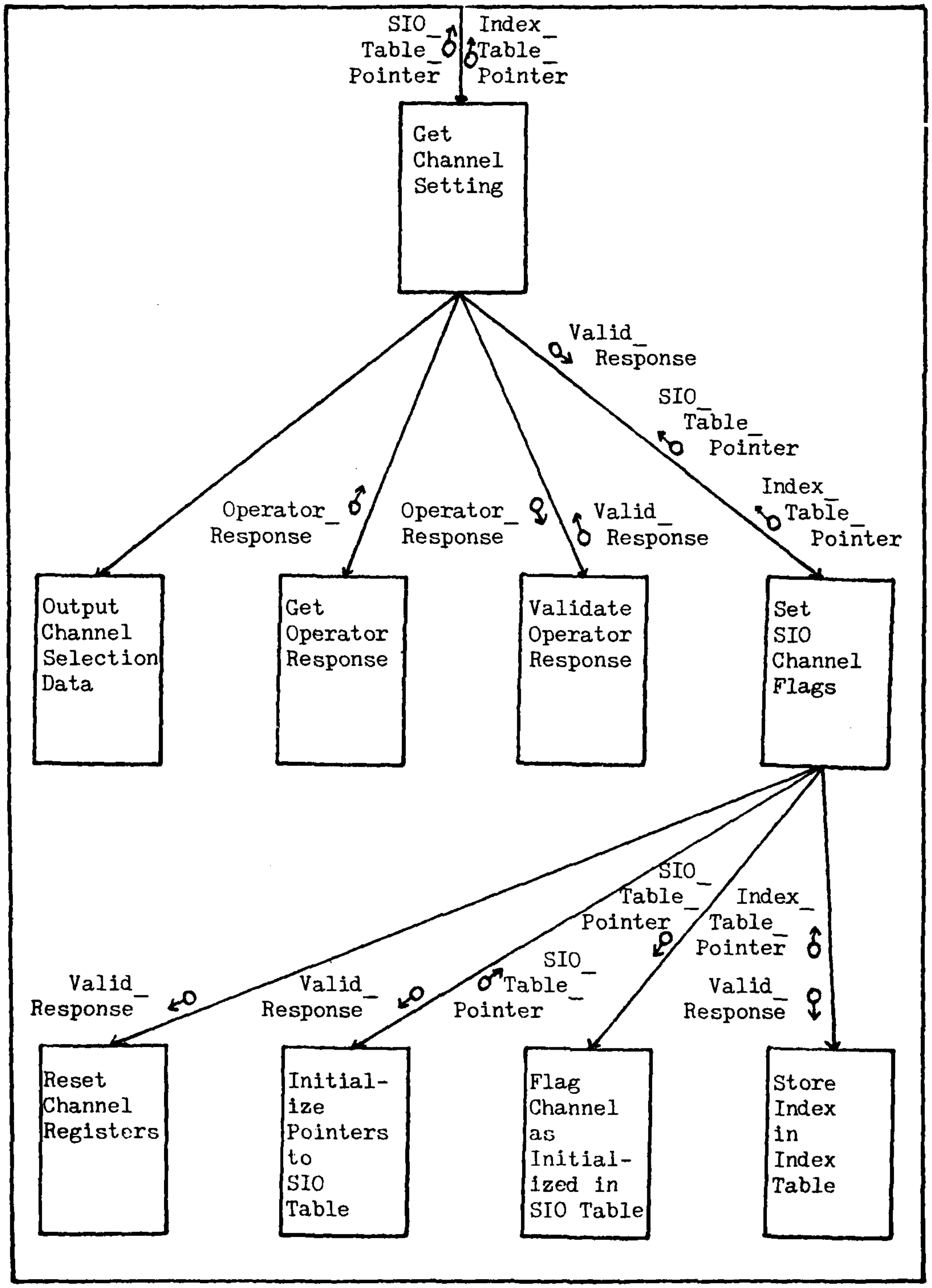
CONIO-INIT--Boot System (Level 1)



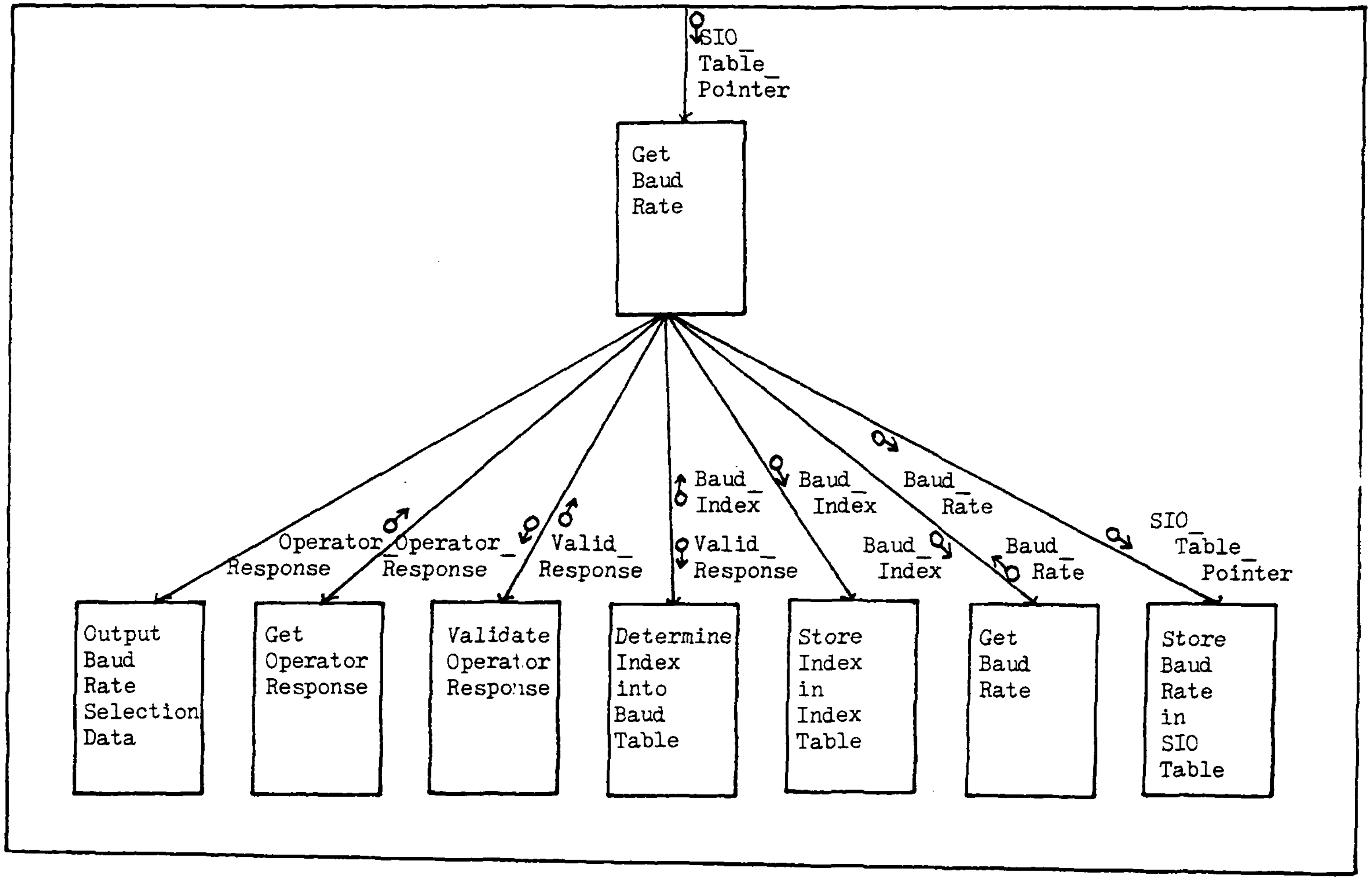
CONIO--Get Operator Input from Console (Level 1)



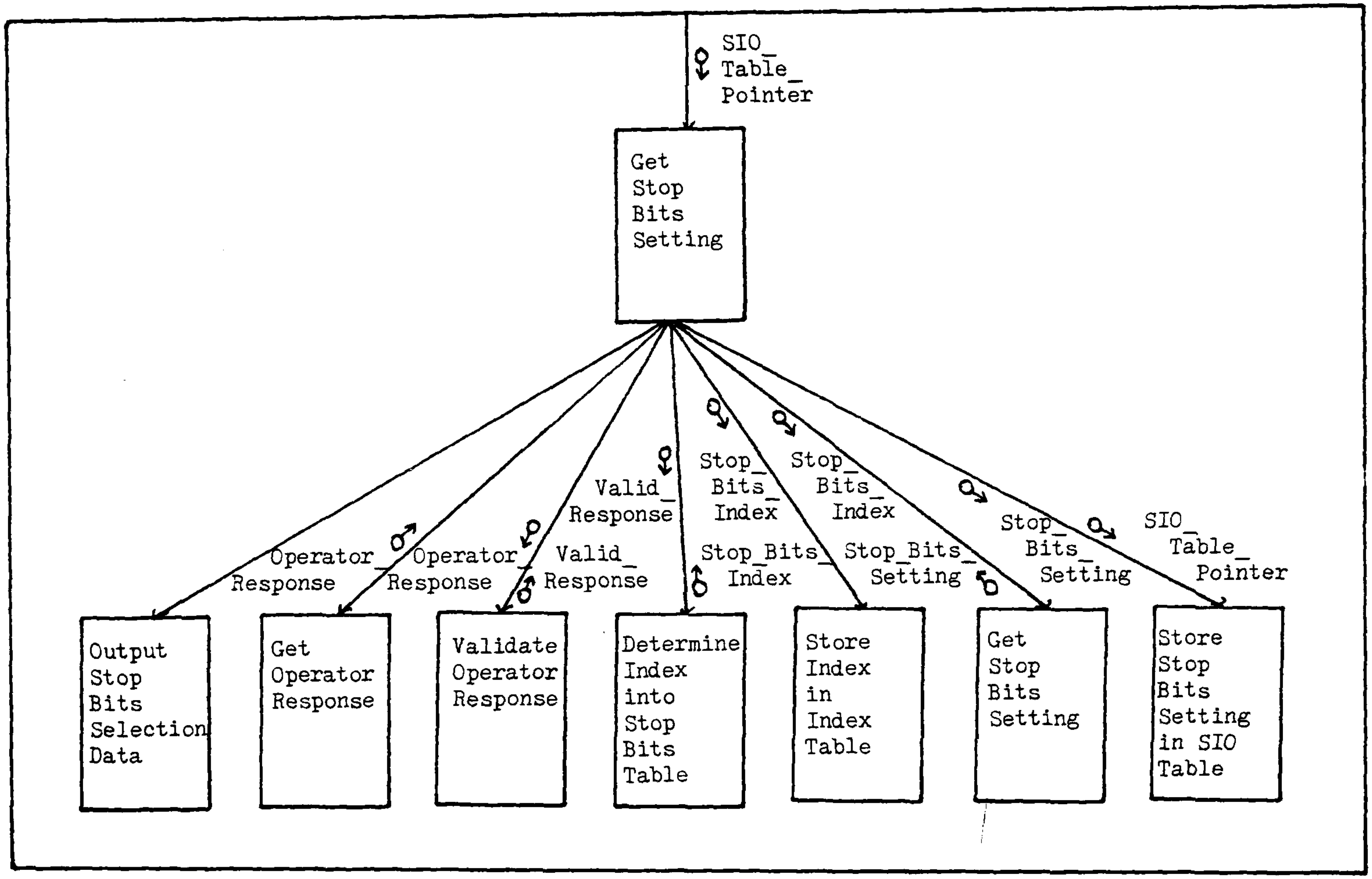
LINE--Design of Line Parameter Sub-Driver (Level 2)



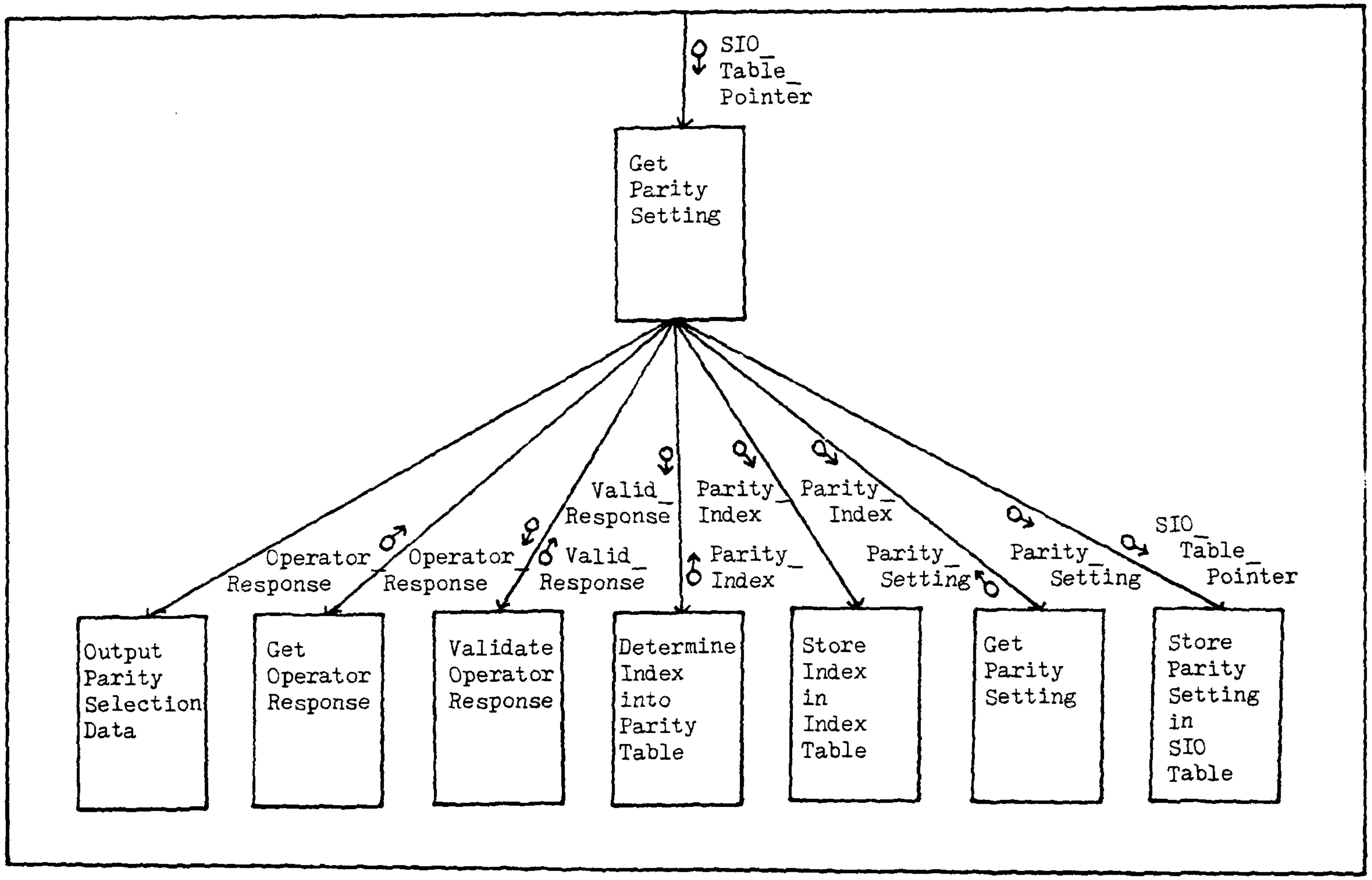
LINE--Get Channel to be Initialized from Operator (Level 3)



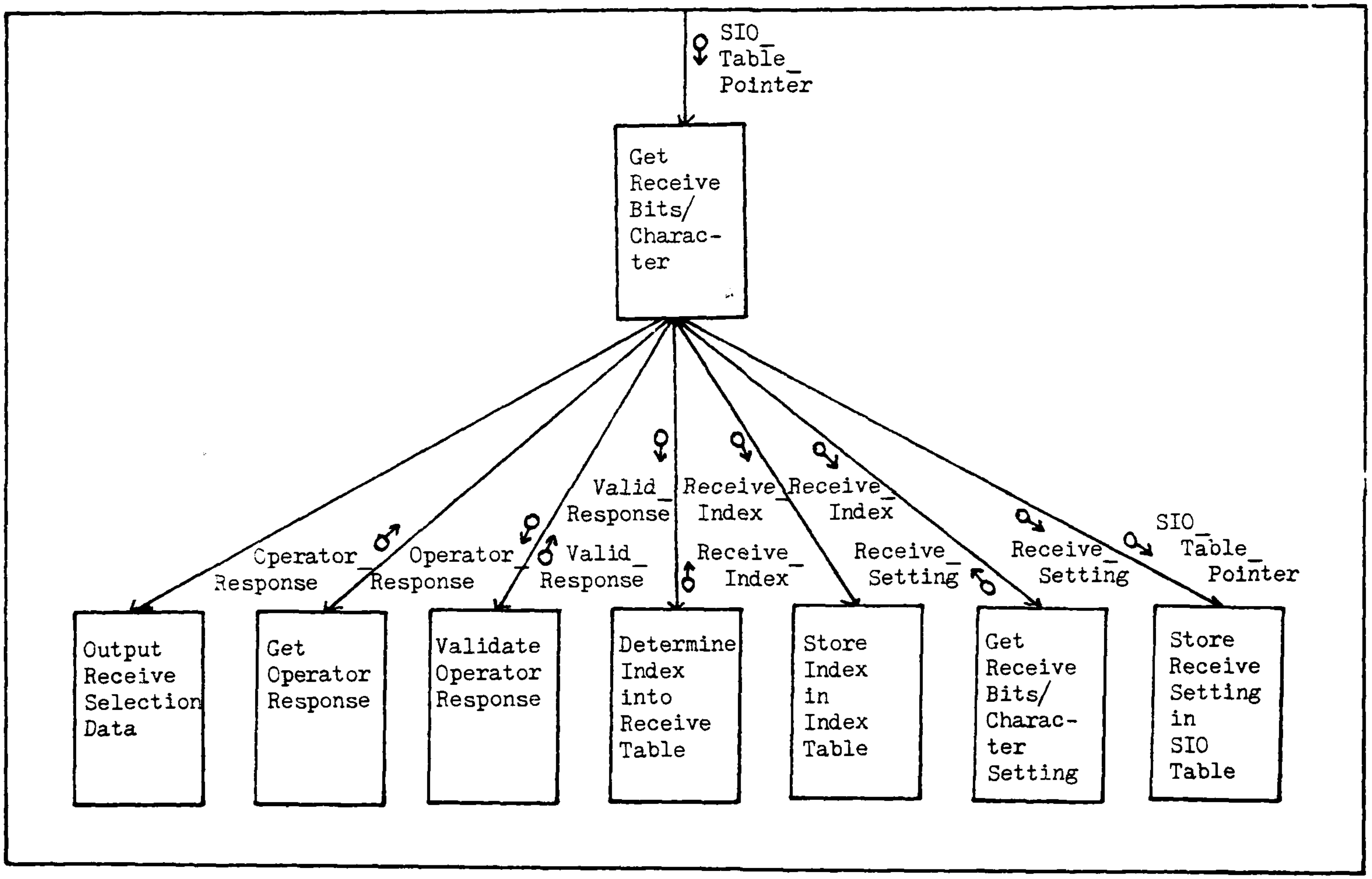
LINE--Get Baud Rate Setting from Operator (Level 3)



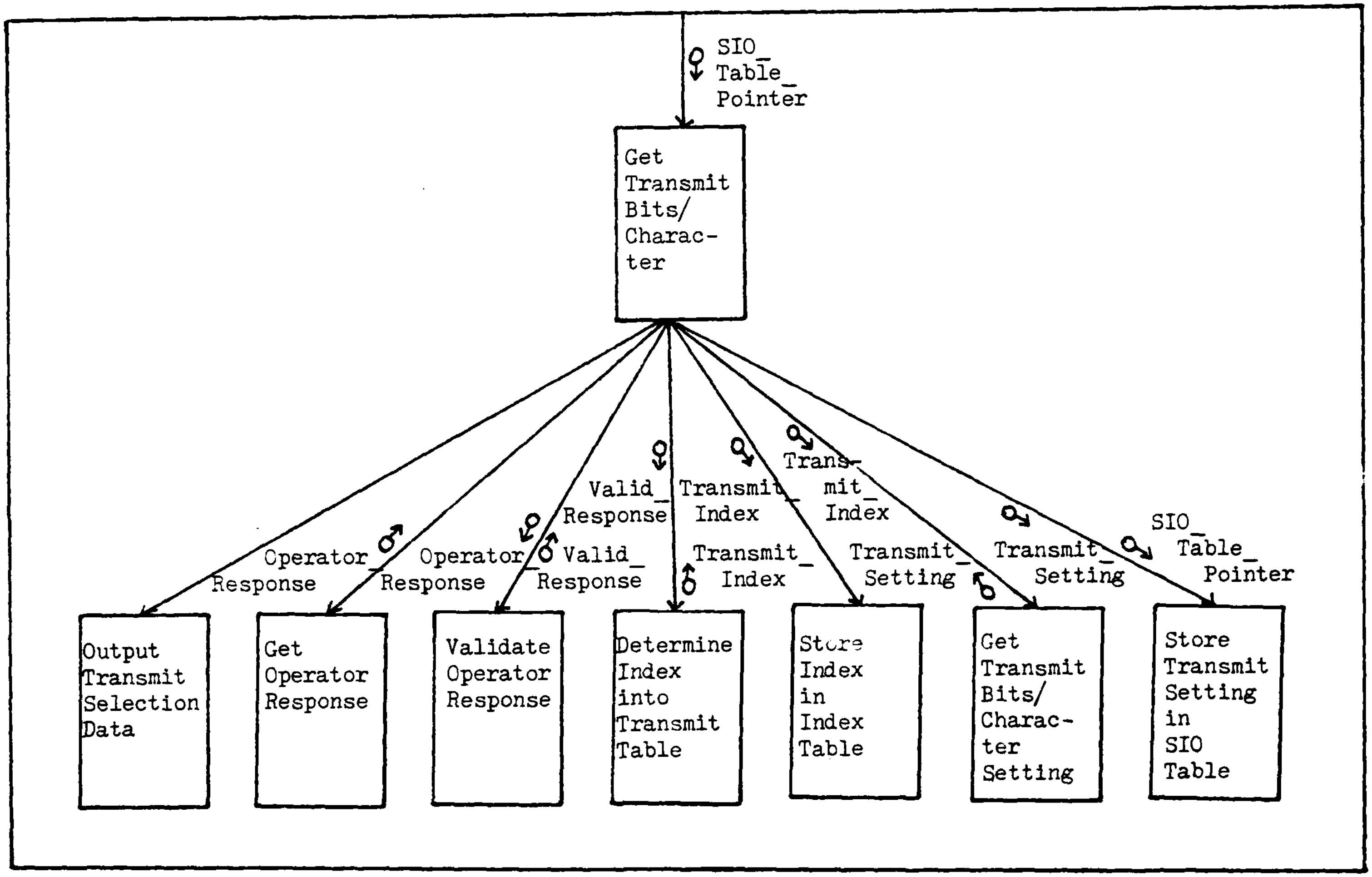
LINE--Get Stop Bits Setting from Operator (Level 3)



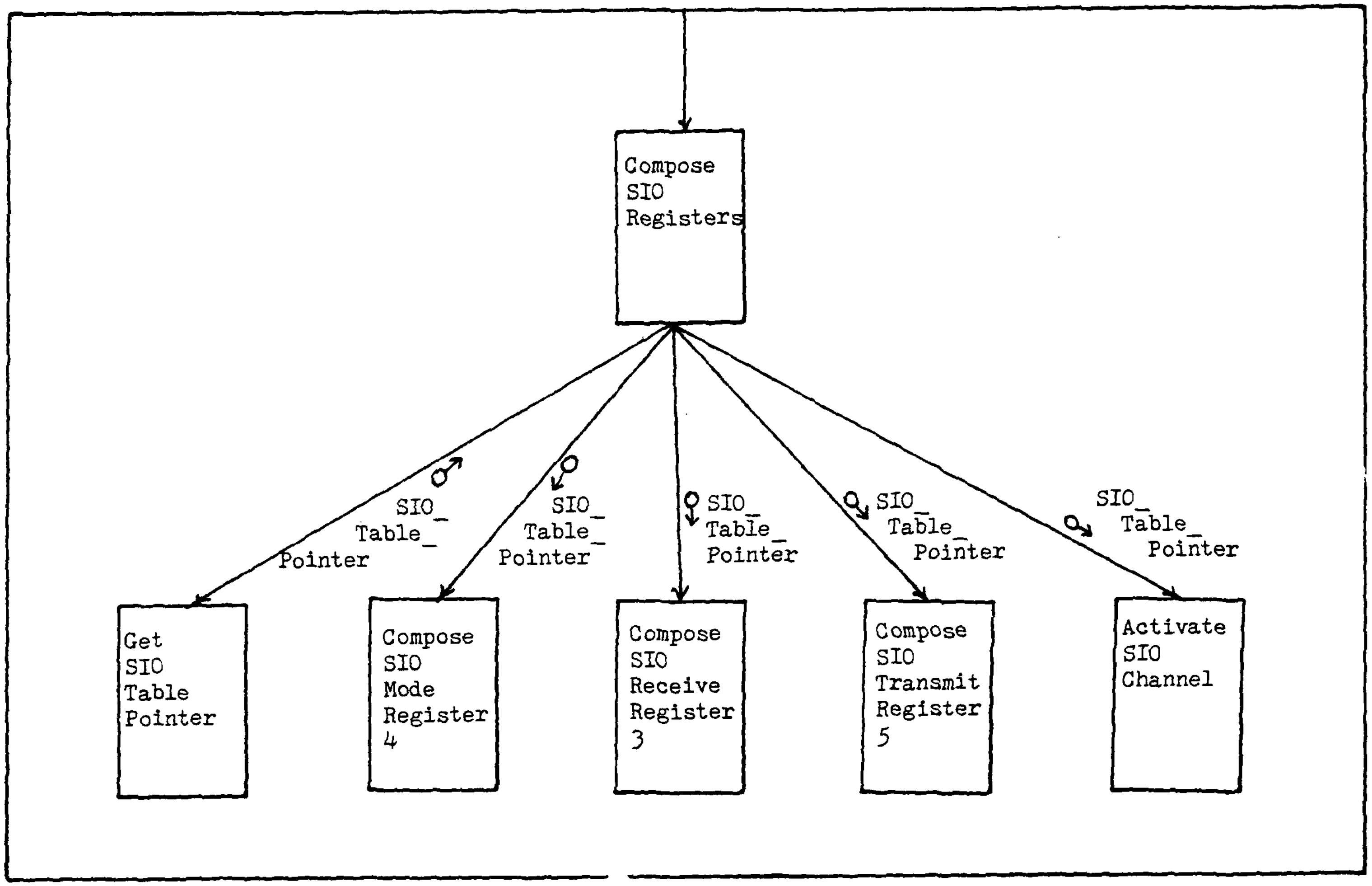
LINE--Get Parity Setting from Operator (Level 3)


SIO_ Table_ Pointer
Get Receive Bits/ Charac- ter
Cperator_ Response
Operator_ Response
Valid_ Response
Valid_ Response
Receive_ Index
Receive_ Index
Receive_ Index
Receive_ Setting
Receive_ Setting
SIO_ Table_ Pointer
Output Receive Selection Data
Get Operator Response
Validate Operator Response
Determine Index into Receive Table
Store Index in Index Table
Get Receive Bits/ Charac- ter Setting
Store Receive Setting in SIO Table

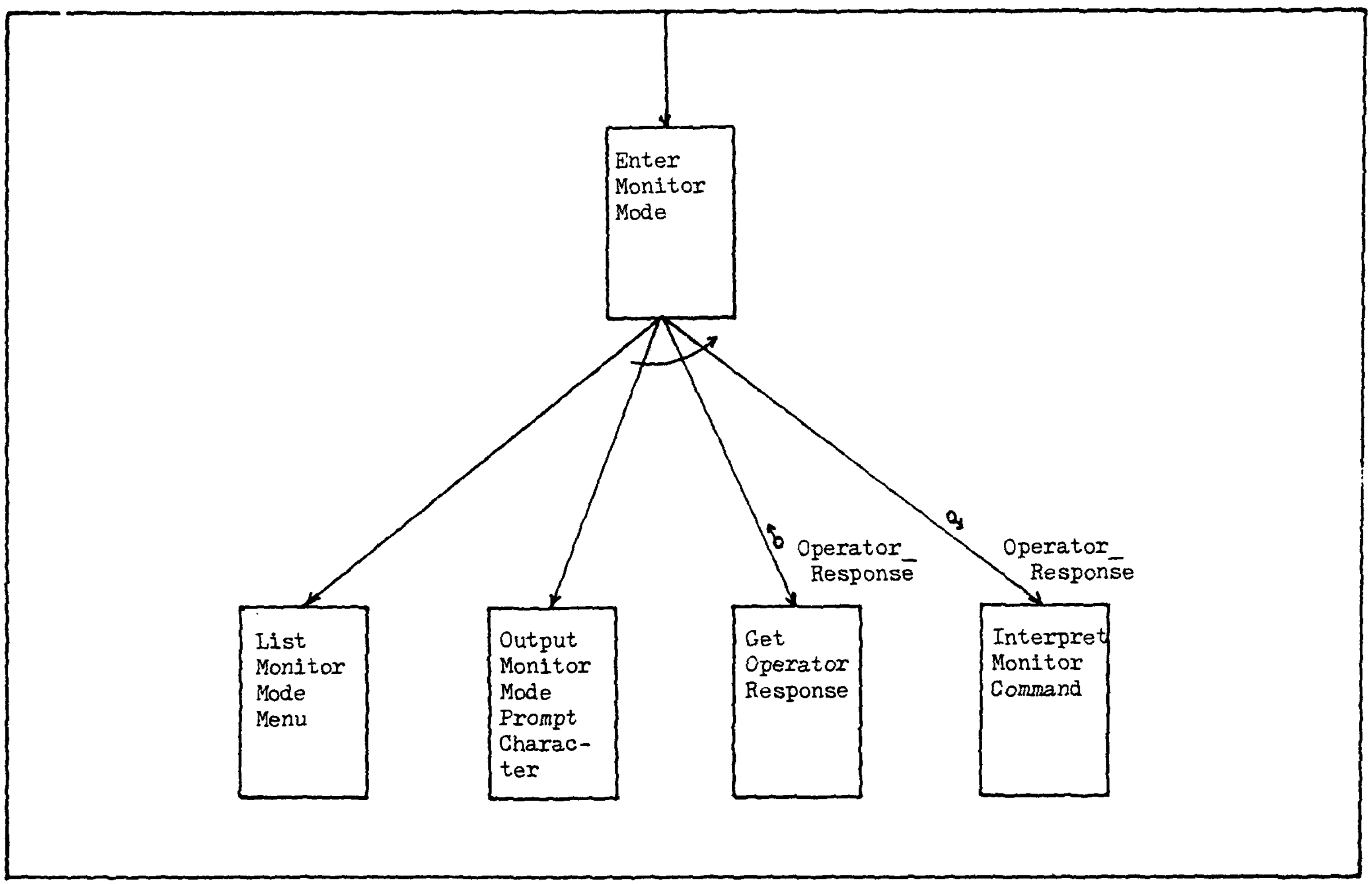
LINE--Get Receive Bits/Character from Operator (Level 3)


SIO_ Table_ Pointer
Get Transmit Bits/ Charac- ter
Operator_ Response
Operator_ Response
Valid_ Response
Valid_ Response
Transmit_ Index
Transmit_ Index
Trans- mit_ Index
Transmit_ Setting
Transmit_ Setting
SIO_ Table_ Pointer
Output Transmit Selection Data
Get Operator Response
Validate Operator Response
Determine Index into Transmit Table
Store Index in Index Table
Get Transmit Bits/ Charac- ter Setting
Store Transmit Setting in SIO Table

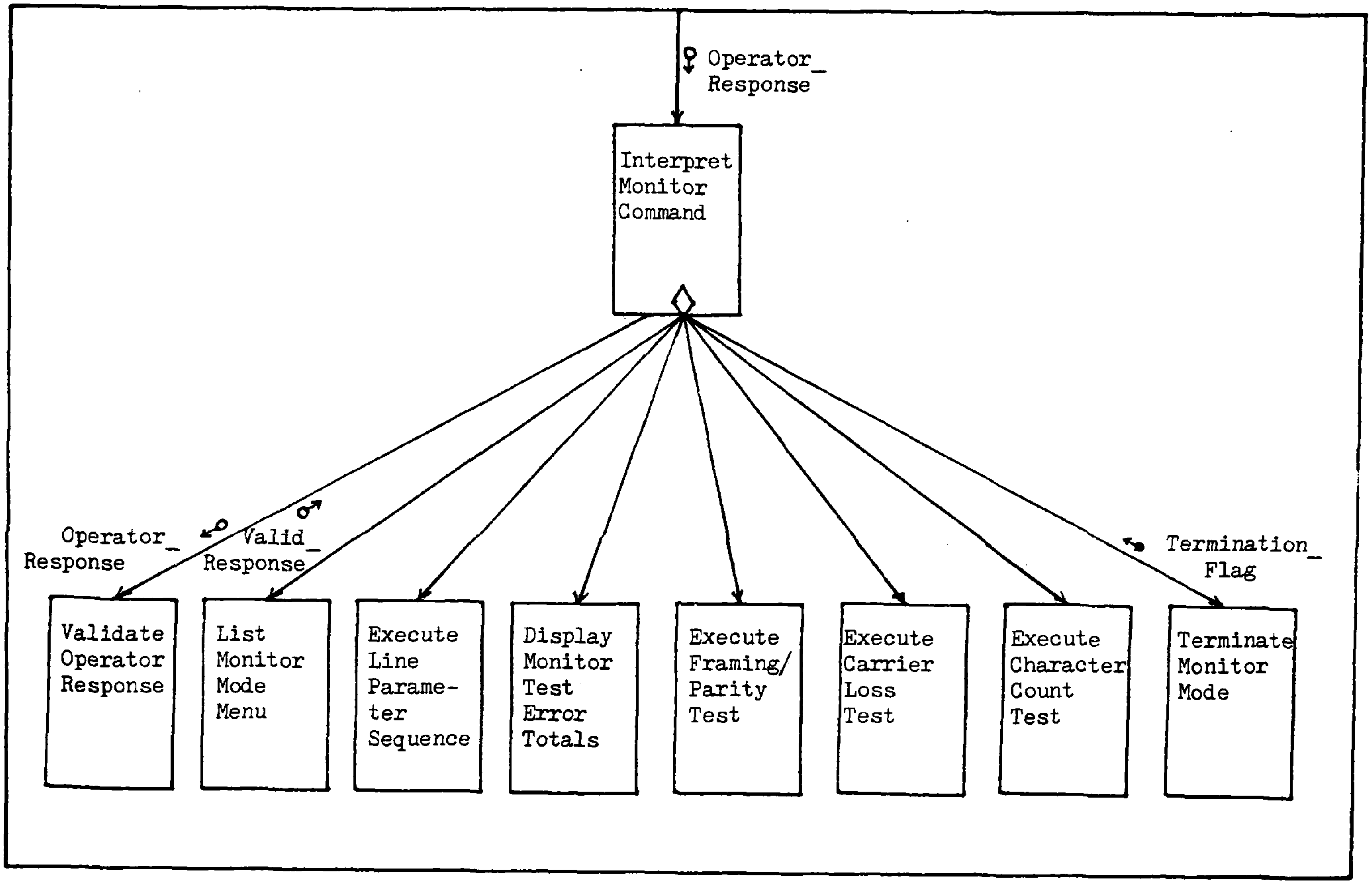
LINE--Get Transmit Bits/Character from Operator (Level 3)



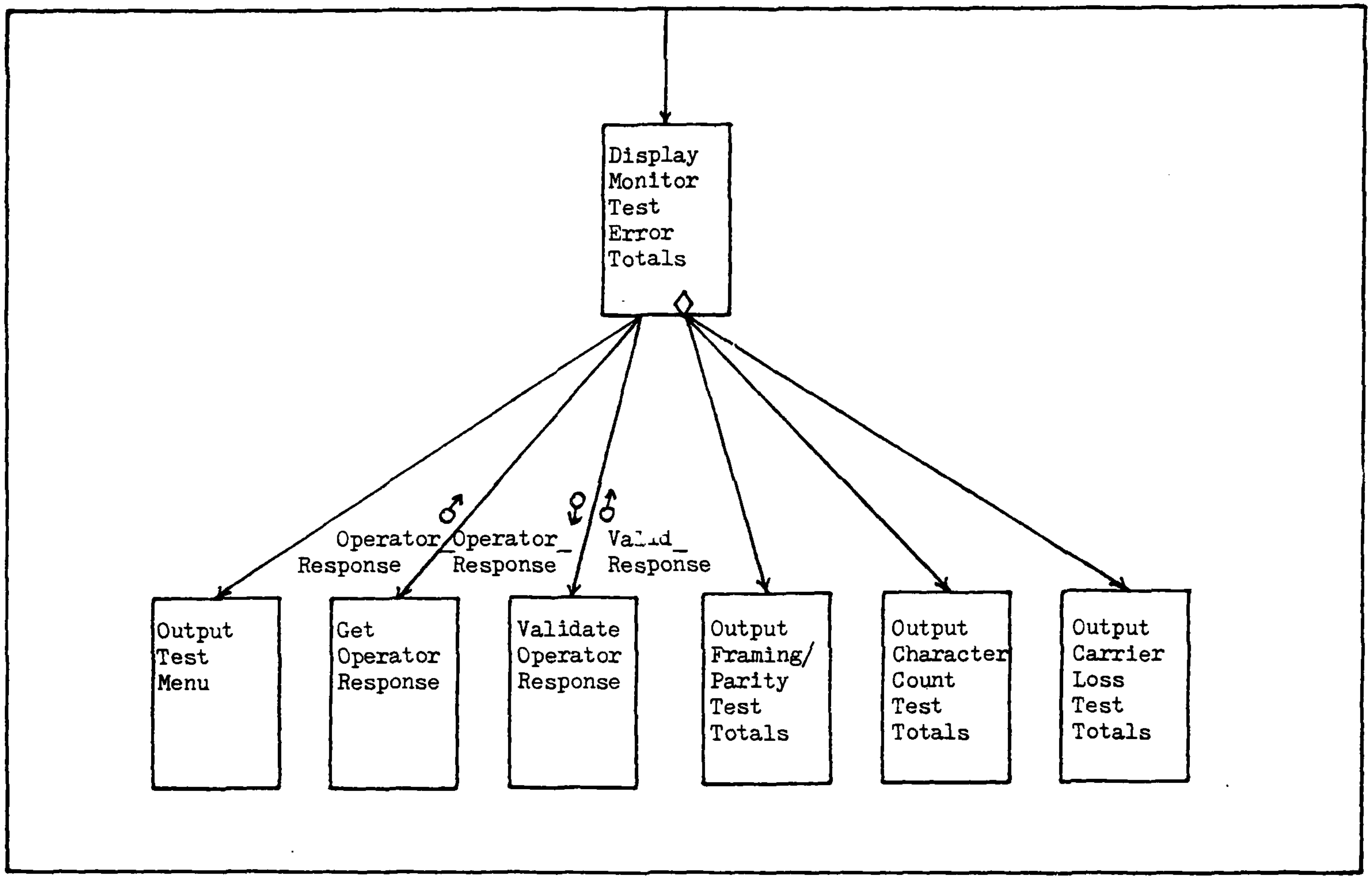
LINE--Using the Parameters Obtained from Operator-Initialize SIO (Level 3)



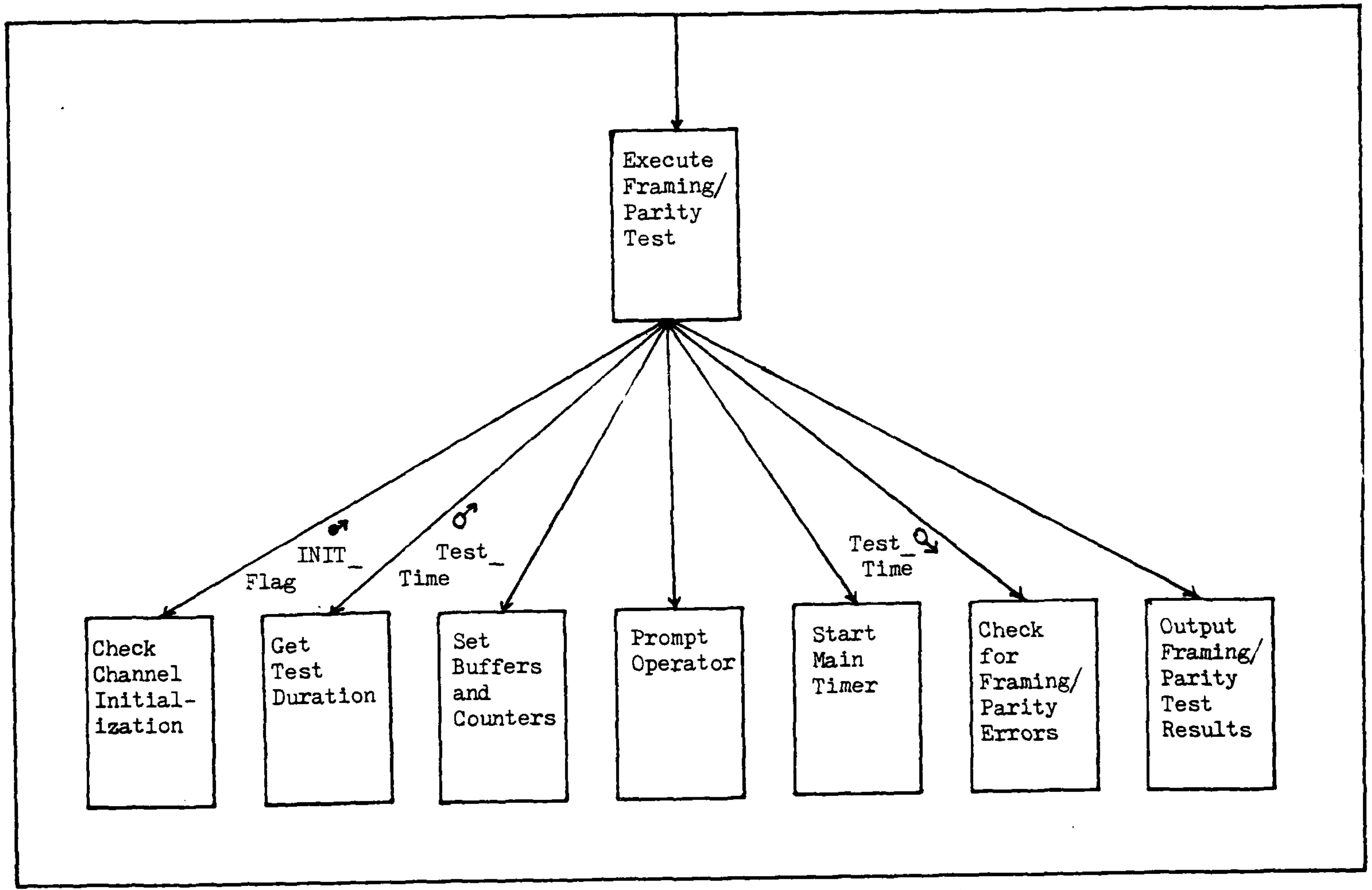
MONITOR--Design of Monitor Mode Sub-Driver (Level 2)



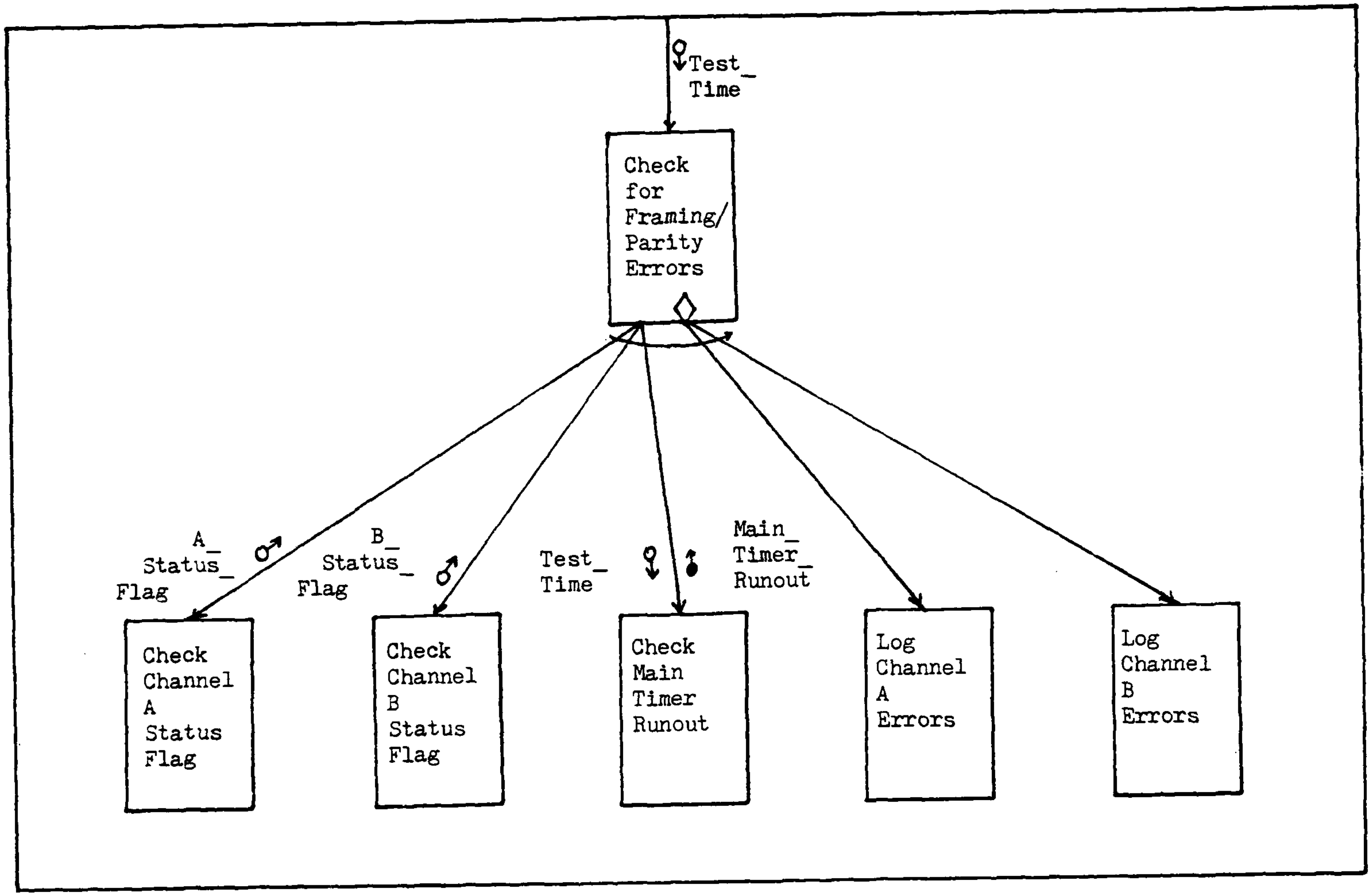
MONITOR--Submodule to Interpret Operator Input at Monitor Mode Level (Level 3)


Display Monitor Test Error Totals
Operator_ Response
Operator_ Response
Valid_ Response
Output Test Menu
Get Operator Response
Validate Operator Response
Output Framing/ Parity Test Totals
Output Character Count Test Totals
Output Carrier Loss Test Totals

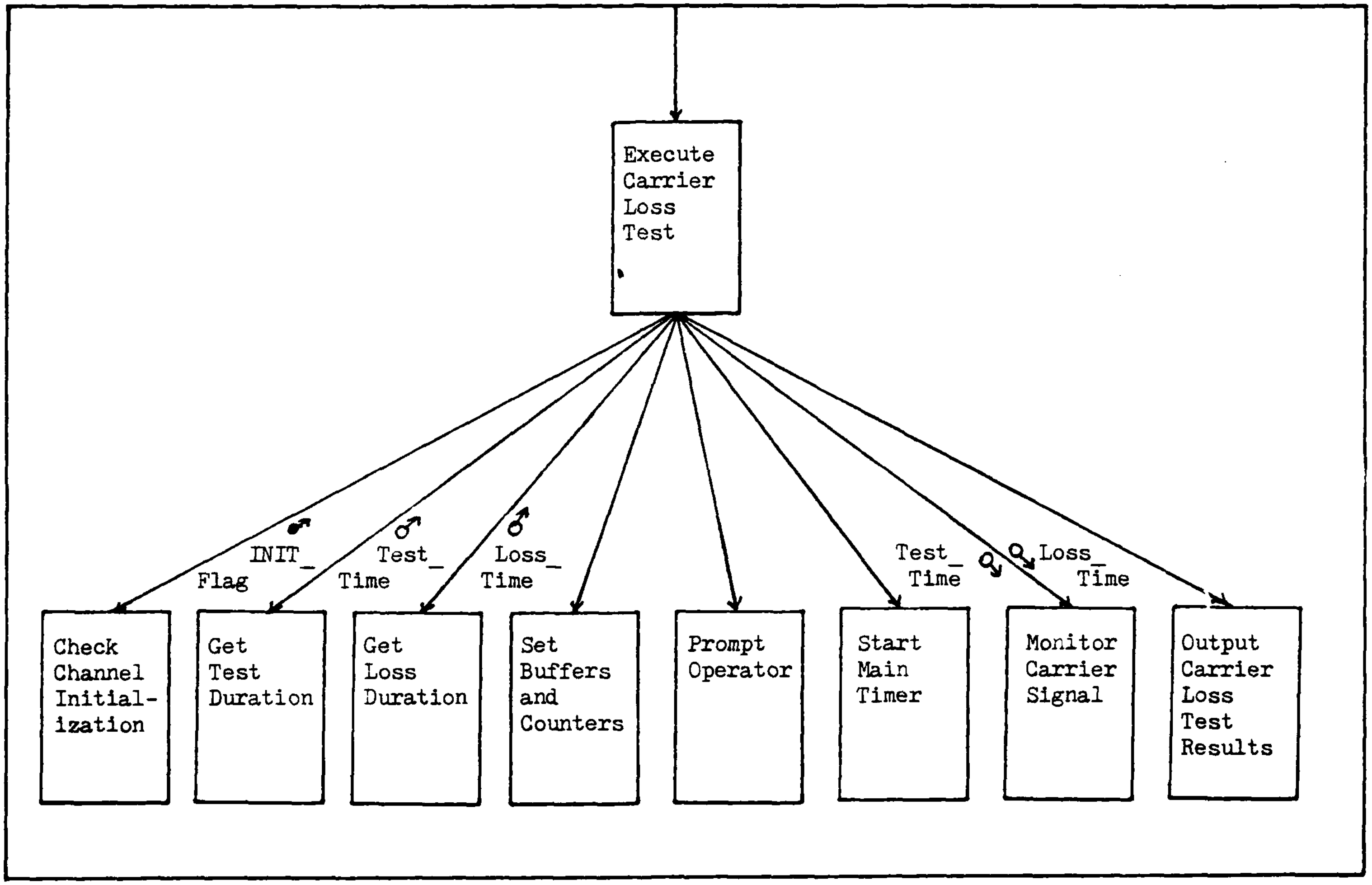
MONITOR--Output Results of Monitor Mode Test (Level 4)


Execute Framing/ Parity Test
INIT_ Flag
Test_ Time
Test_ Time
Check Channel Initial- ization
Get Test Duration
Set Buffers and Counters
Prompt Operator
Start Main Timer
Check for Framing/ Parity Errors
Output Framing/ Parity Test Results

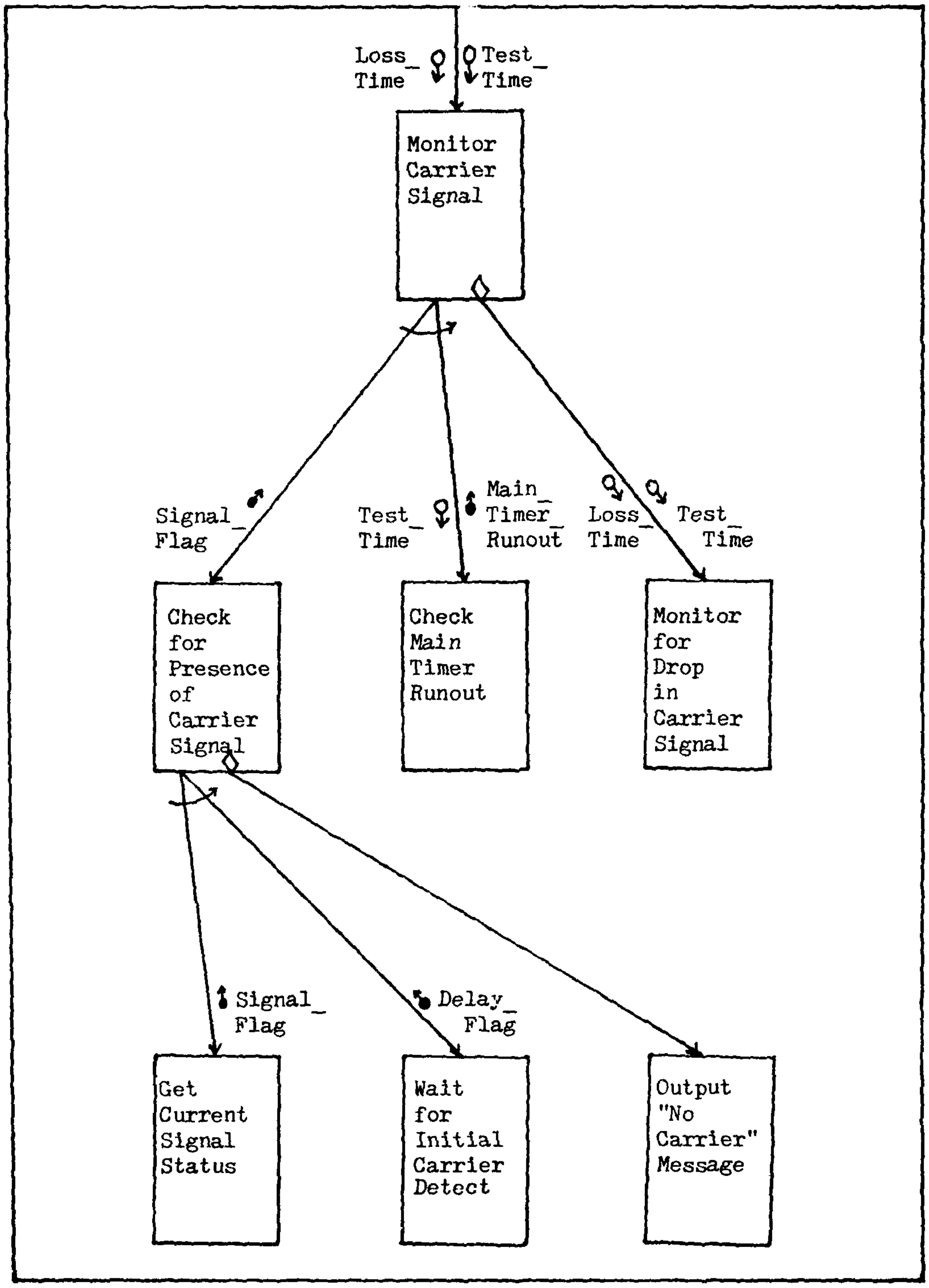
MONITOR--Execute Framing/Parity Test (Level 4)



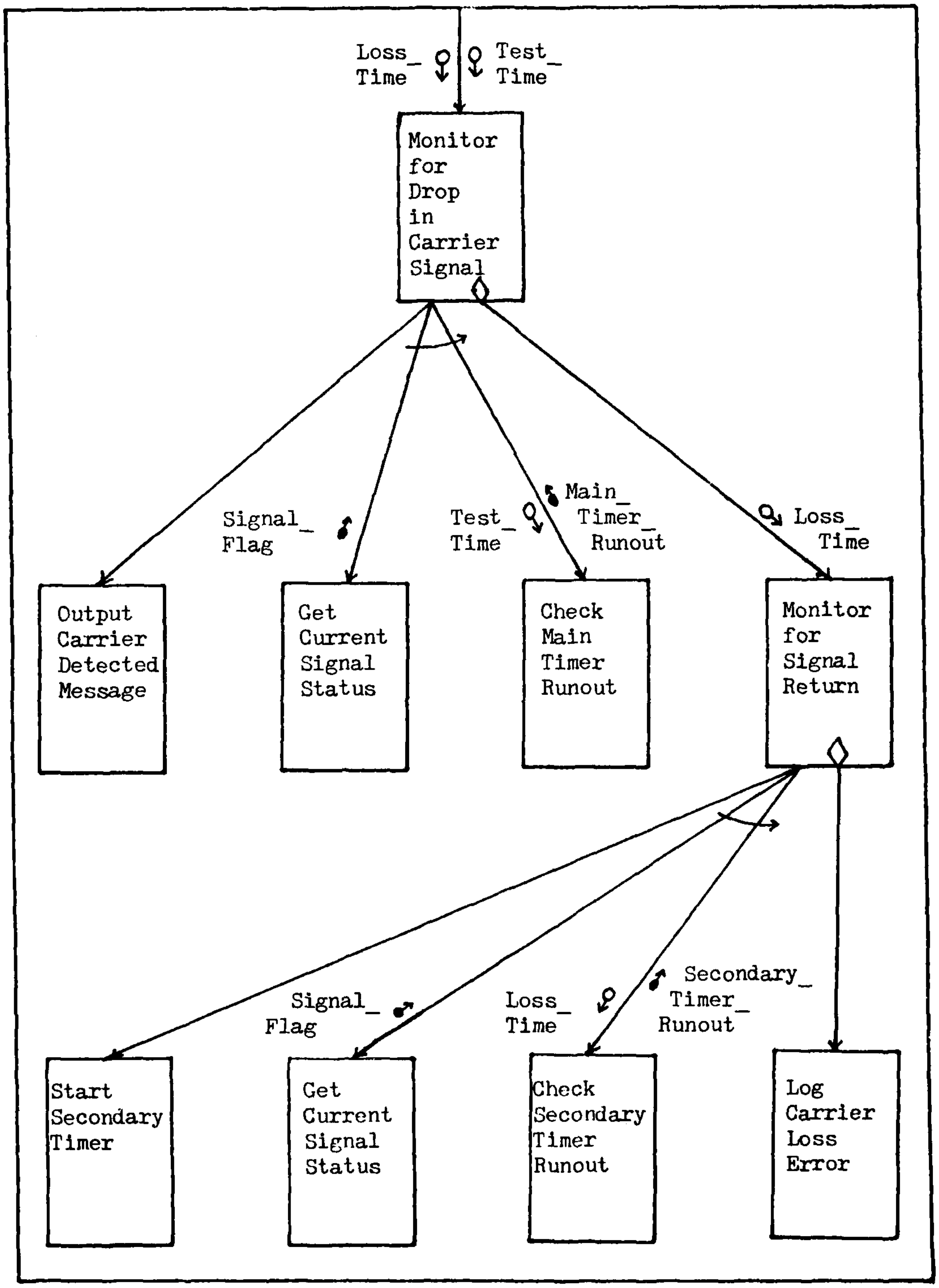
MONITOR--Check for Framing/Parity Errors (Level 5)



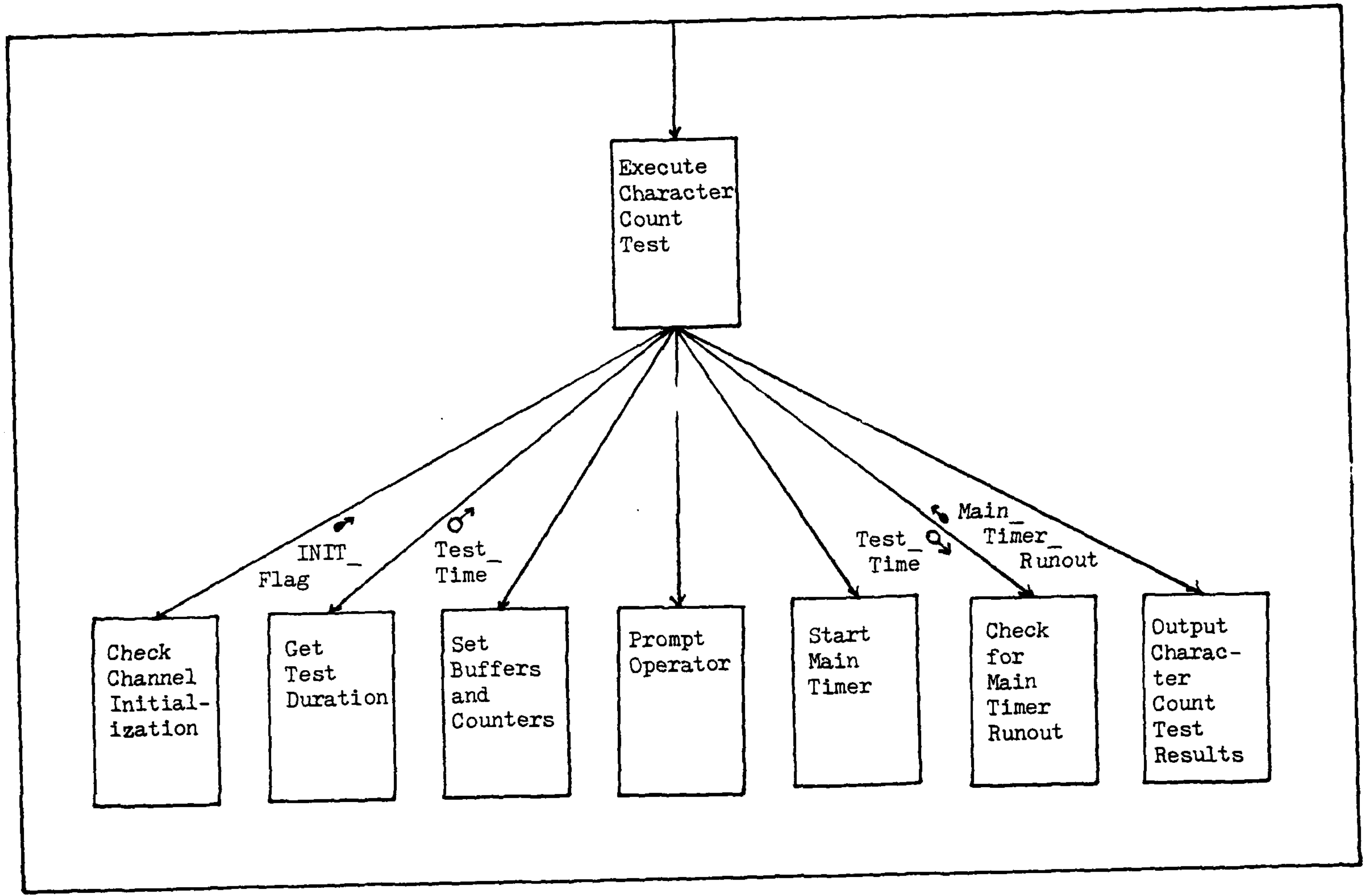
MONITOR--Execute Carrier Loss Test (Level 4)


Loss_ Time
Test_ Time
Monitor Carrier Signal
Signal_ Flag
Test_ Time
Main_ Timer_ Runout
Loss_ Time
Test_ Time
Check for Presence of Carrier Signal
Check Main Timer Runout
Monitor for Drop in Carrier Signal
Signal_ Flag
Delay_ Flag
Get Current Signal Status
Wait for Initial Carrier Detect
Output "No Carrier" Message

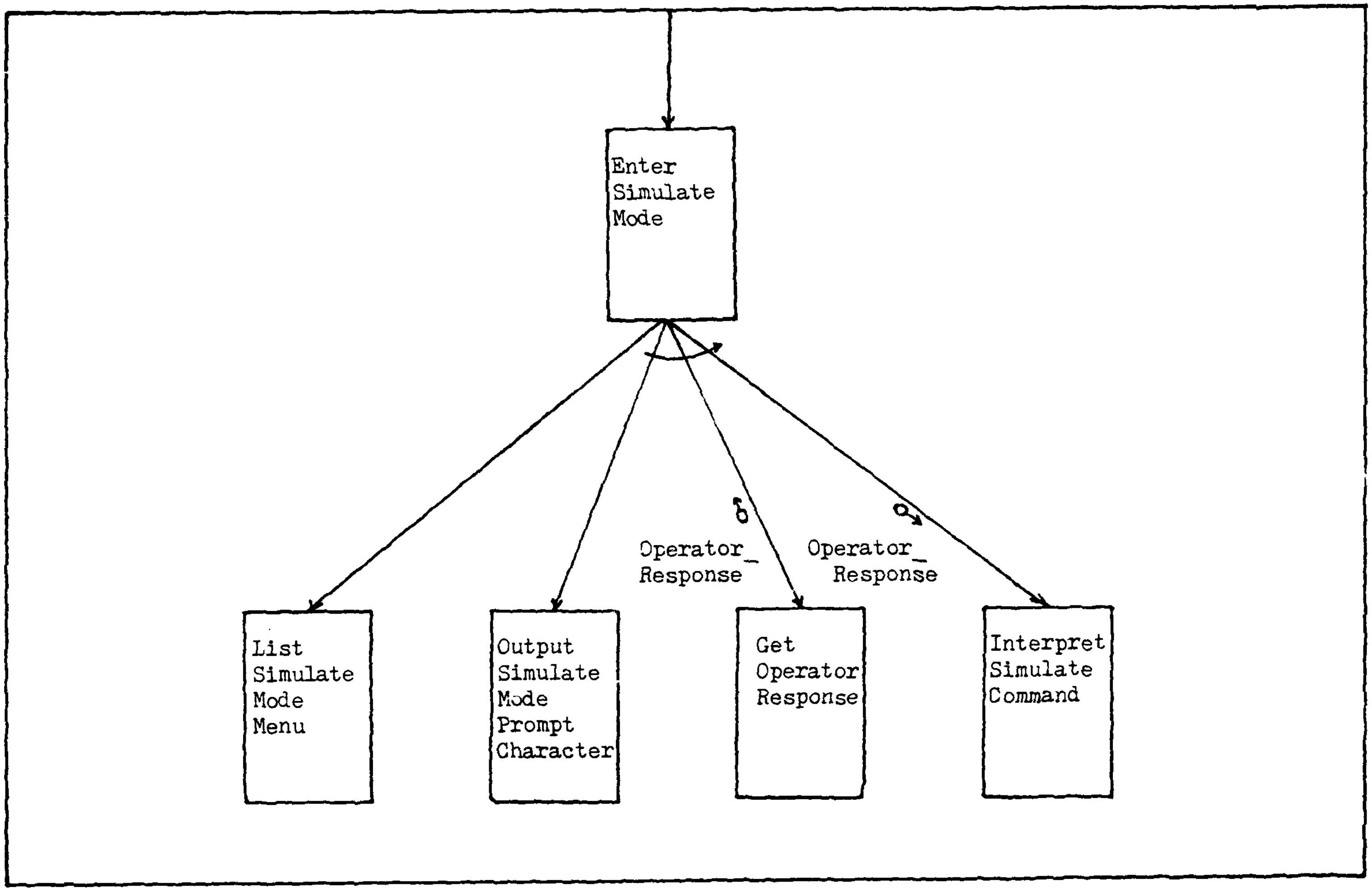
MONITOR--Monitor Carrier Signal (Level 5)



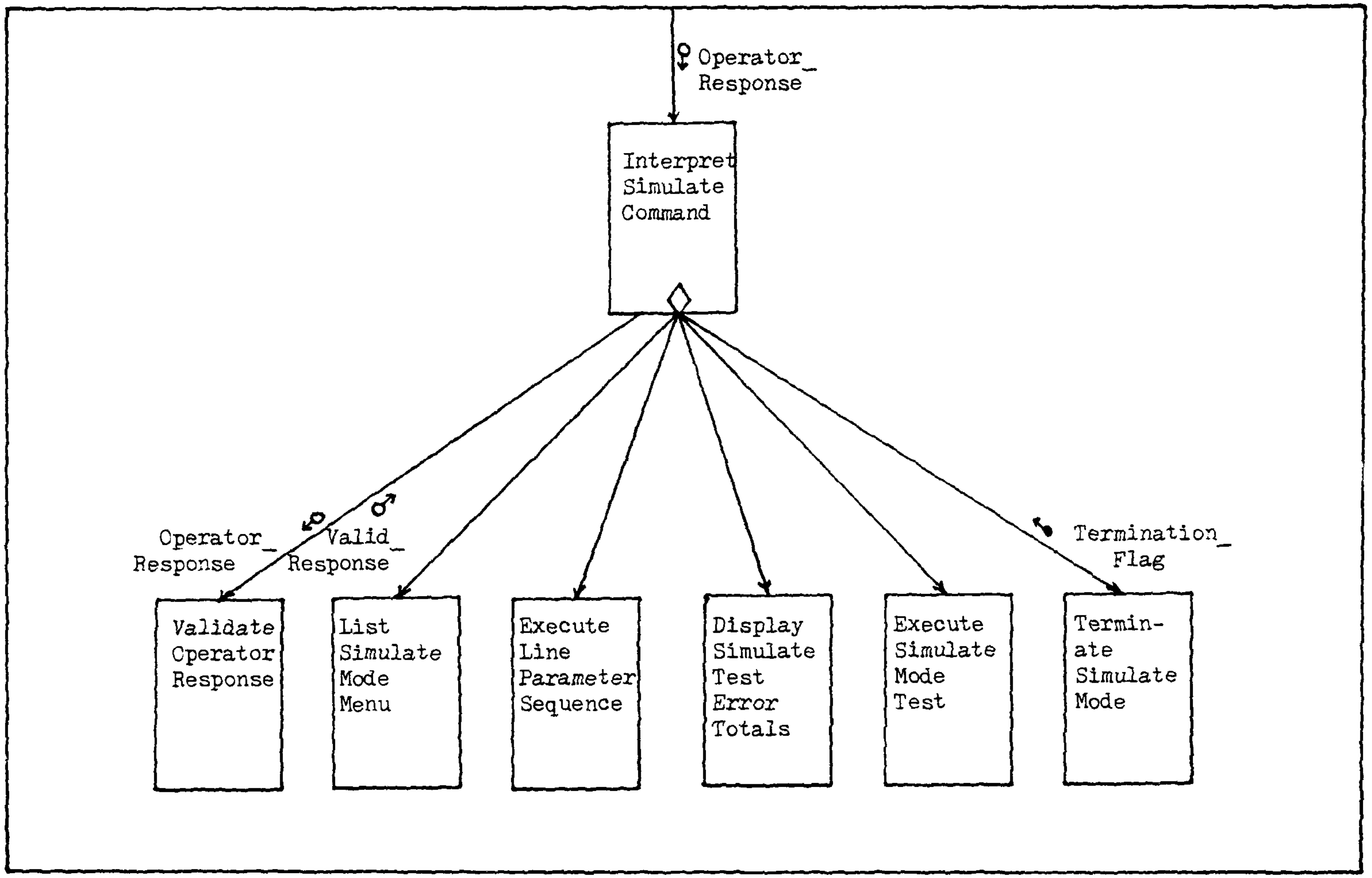
MONITOR--Monitor for Drop in Carrier Signal (Level 6)


Execute Character Count Test
INIT_ Flag
Test_ Time
Test_ Time
Main_ Timer_ Runout
Check Channel Initial- ization
Get Test Duration
Set Buffers and Counters
Prompt Operator
Start Main Timer
Check for Main Timer Runout
Output Charac- ter Count Test Results

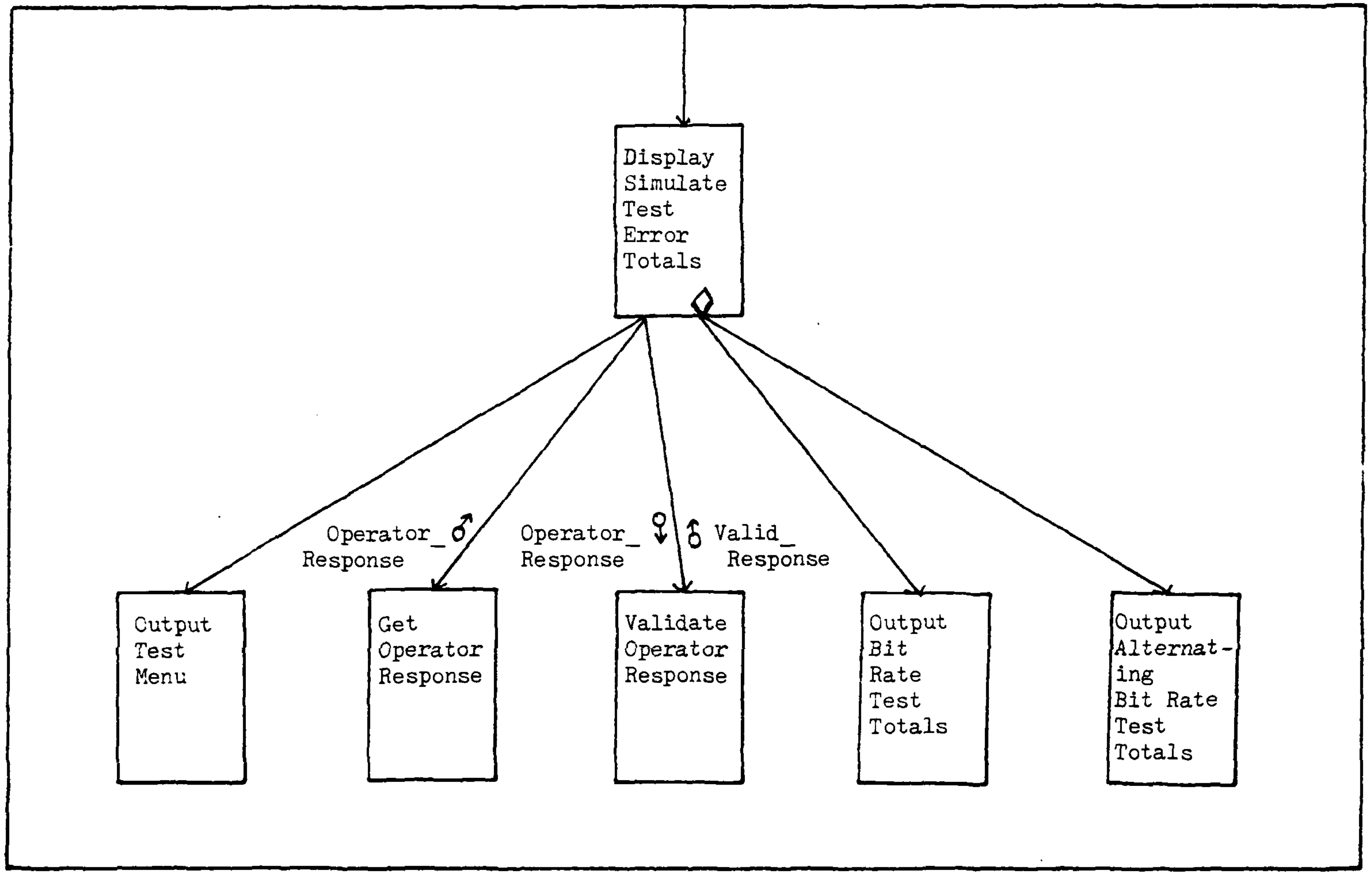
MONITOR--Execute Character Count Test (Level 4)



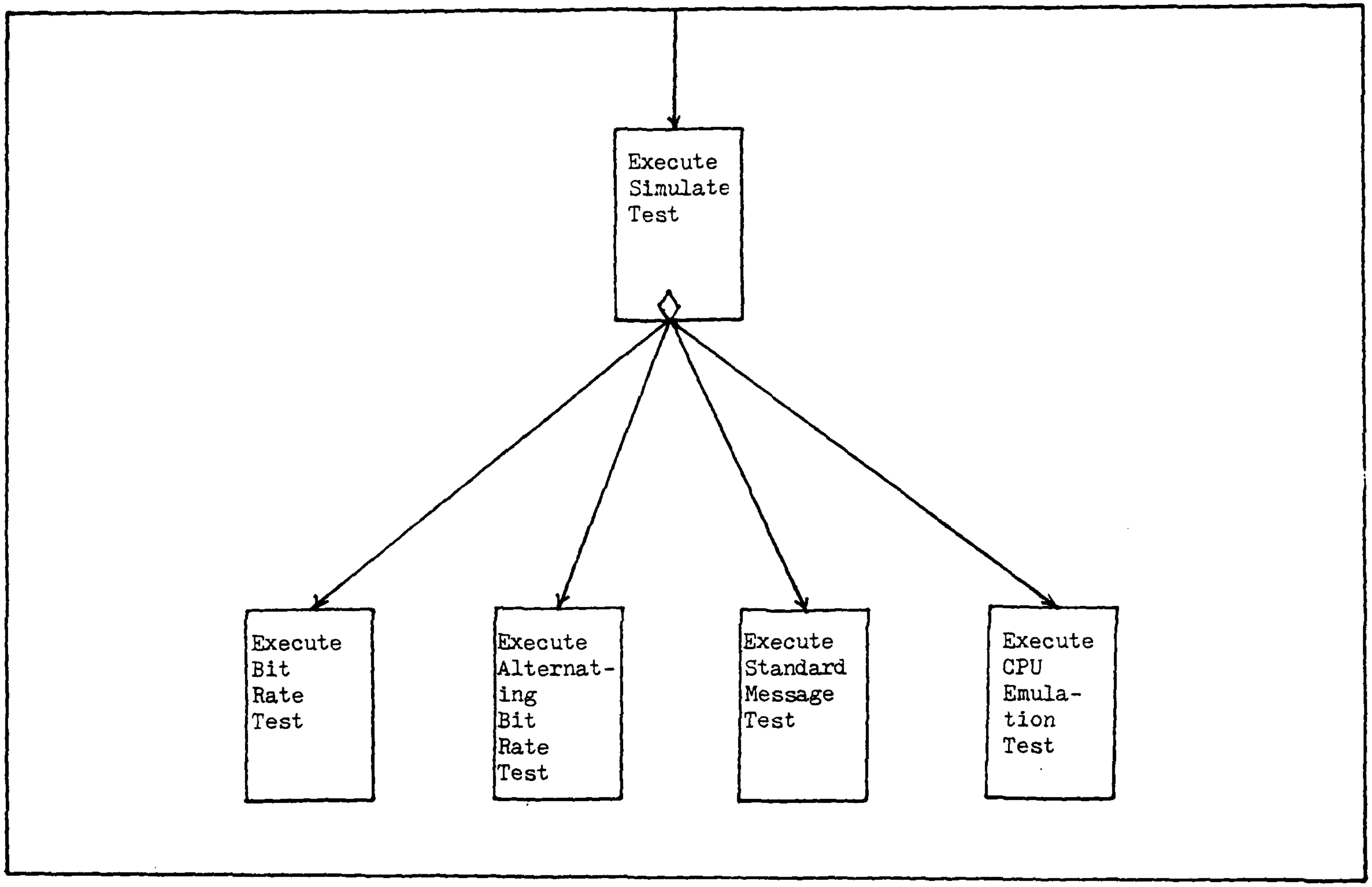
SIMULATE--Design of Simulate Mode Sub-Driver (Level 2)



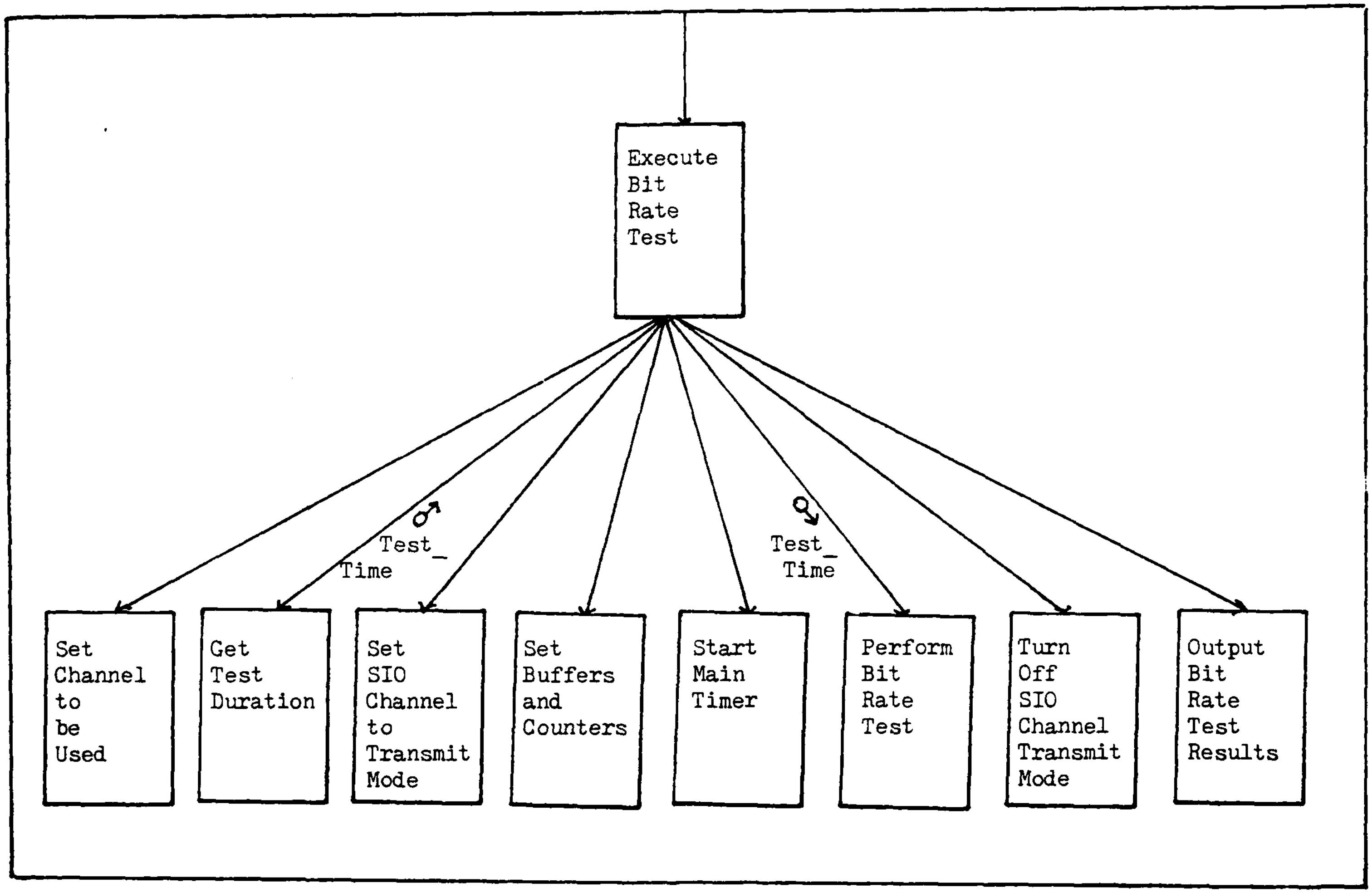
SIMULATE--Submodule to Interpret Operator Input at Simulate Mode Level (Level 3)



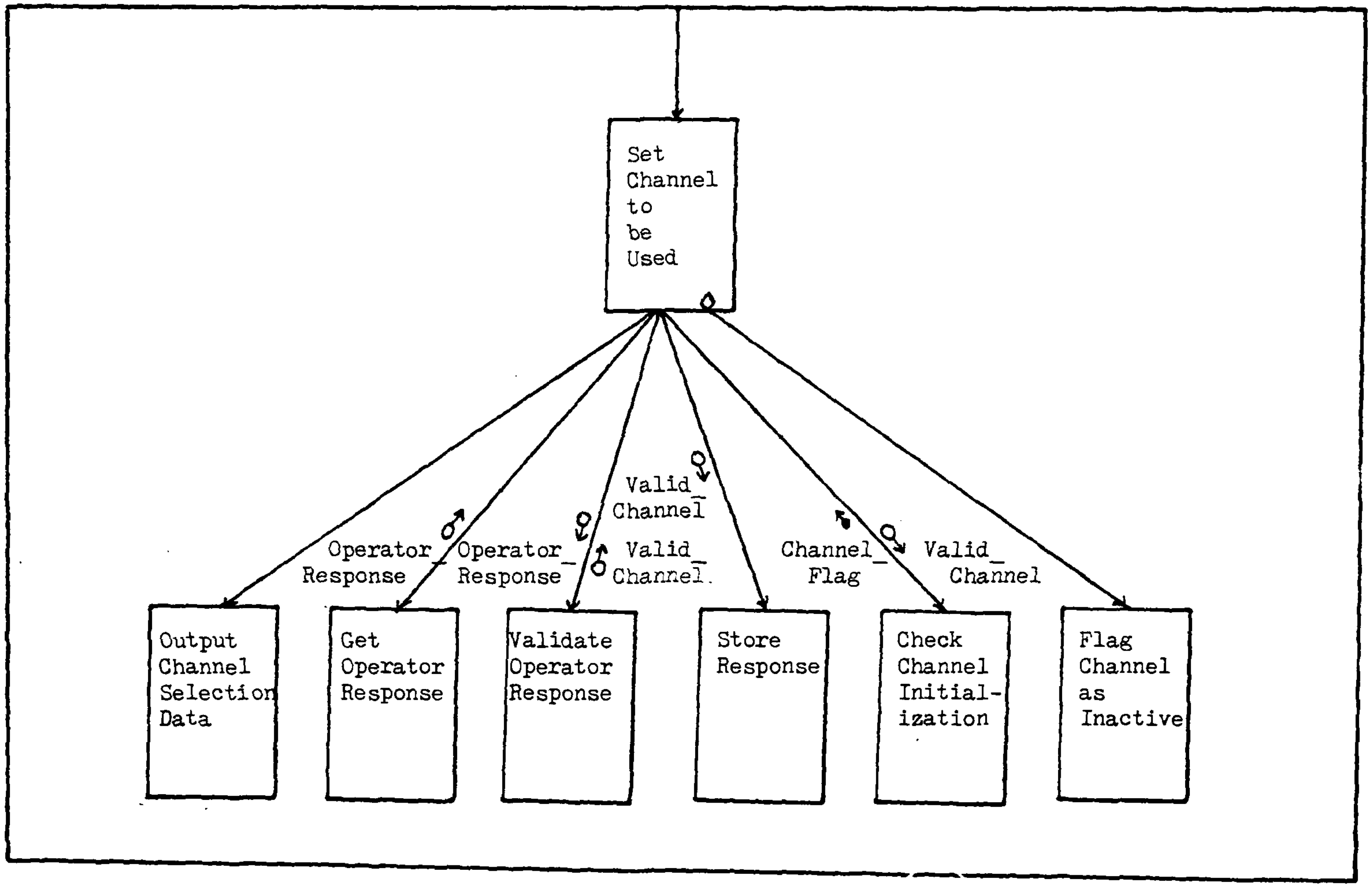
SIMULATE-- Output Simulate Test Results (Level 4)


Execute Simulate Test
Execute Bit Rate Test
Execute Alternating Bit Rate Test
Execute Standard Message Test
Execute CPU Emulation Test

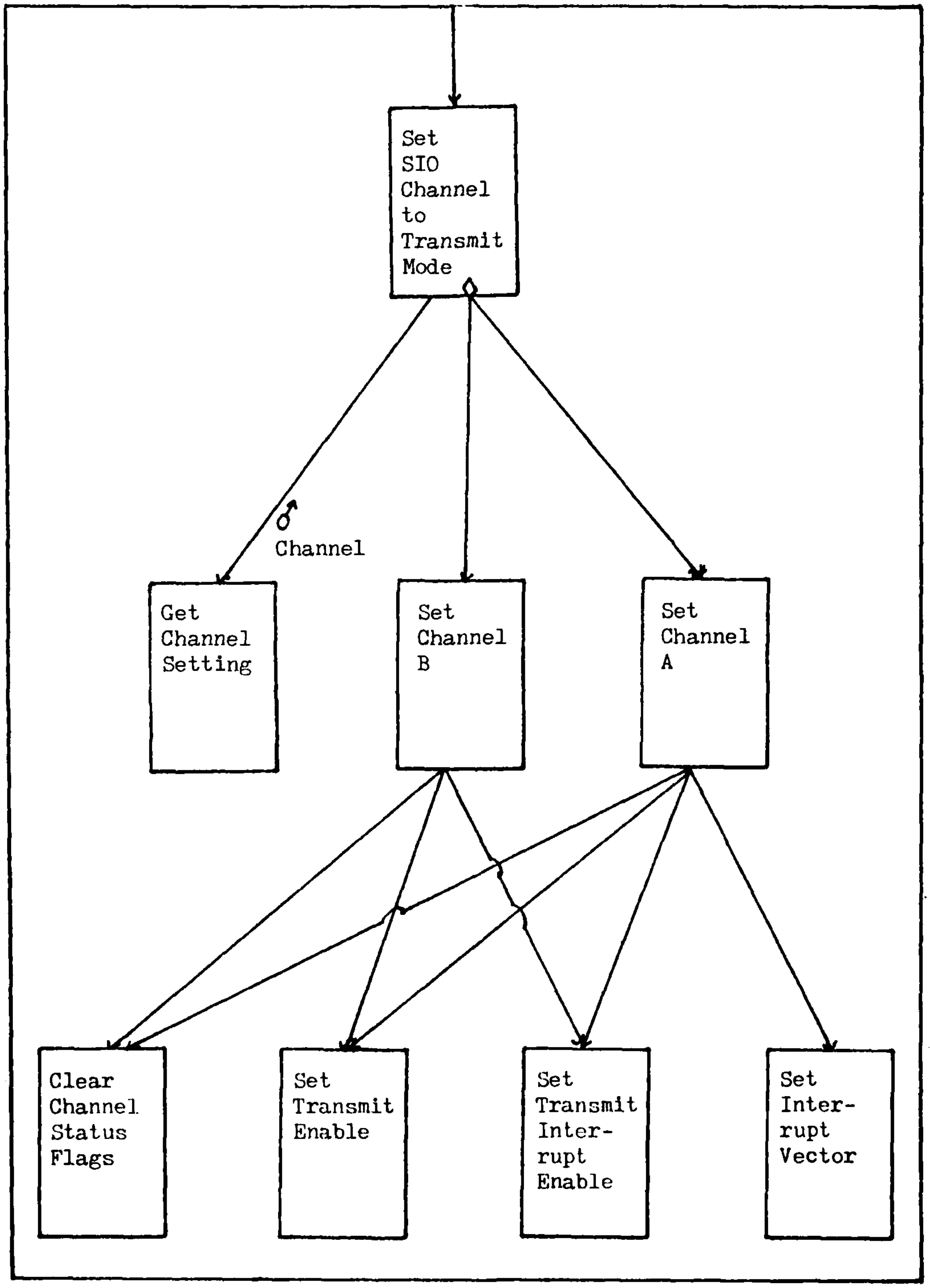
SIMULATE--Execute Simulate Mode Test (Level 4)



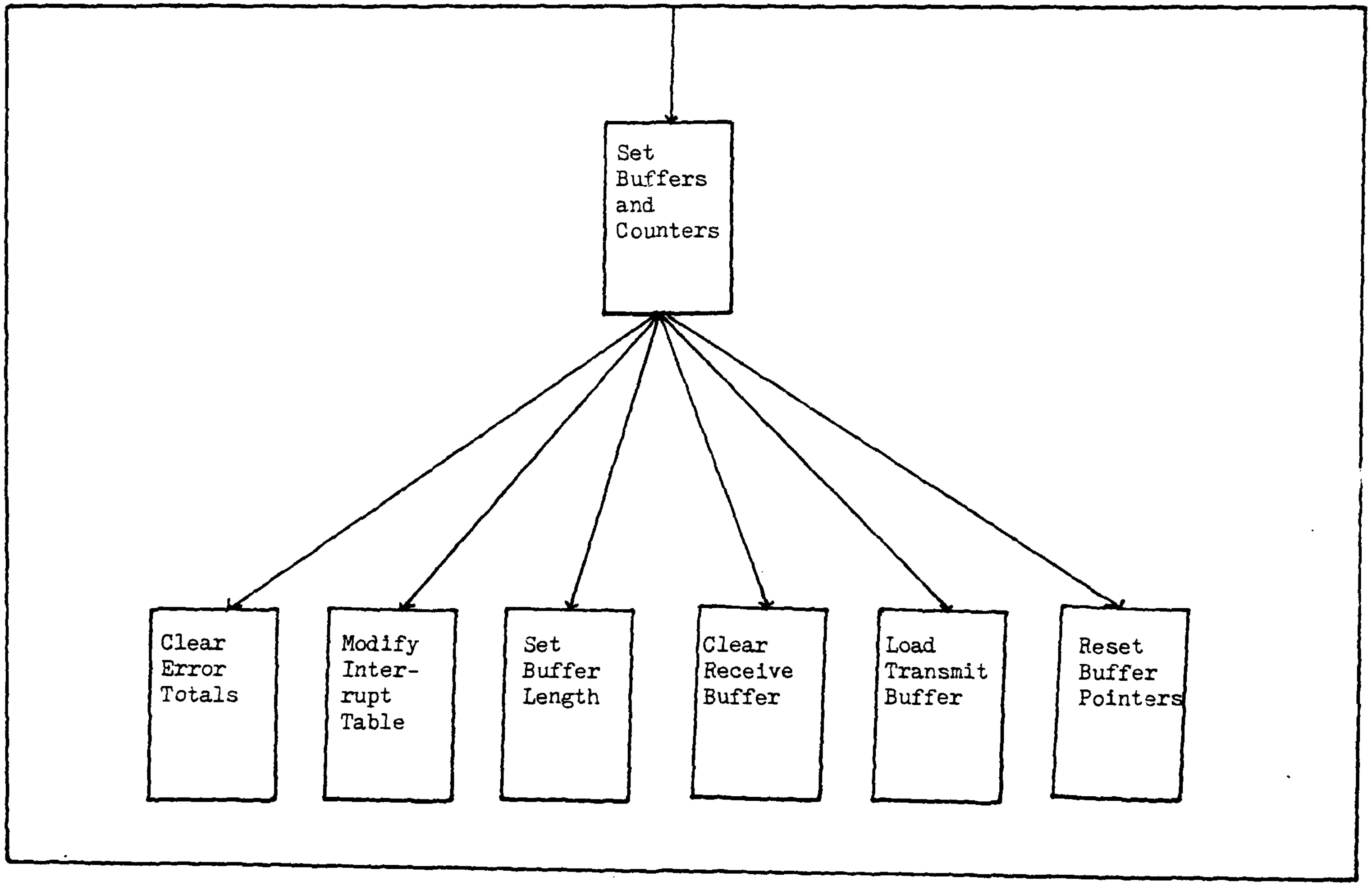
SIMULATE--Execute Bit Rate Test (Level 4)



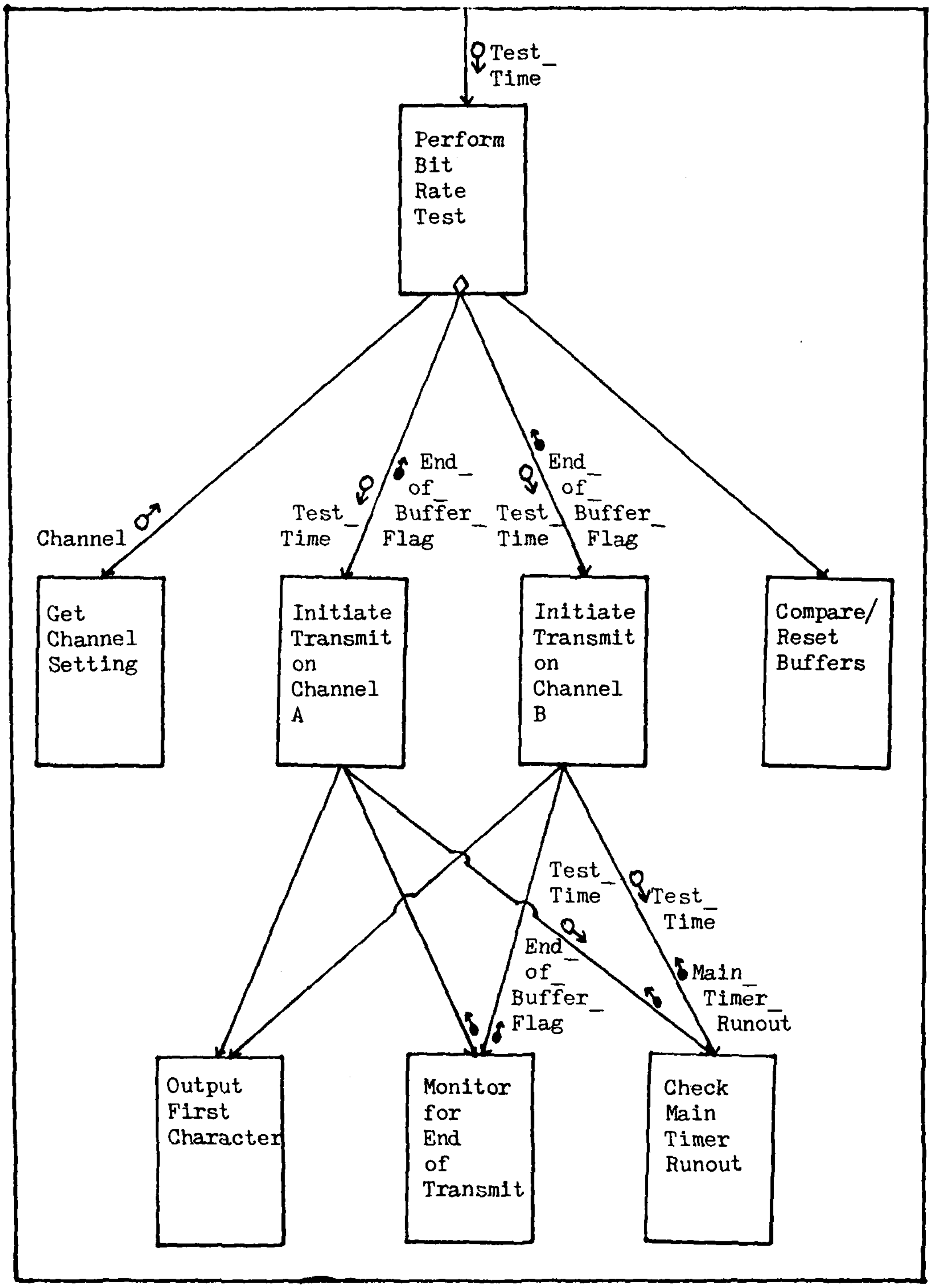
SIMULATE--Initialize the Channel to be Used in Simulation Test (Level 5)



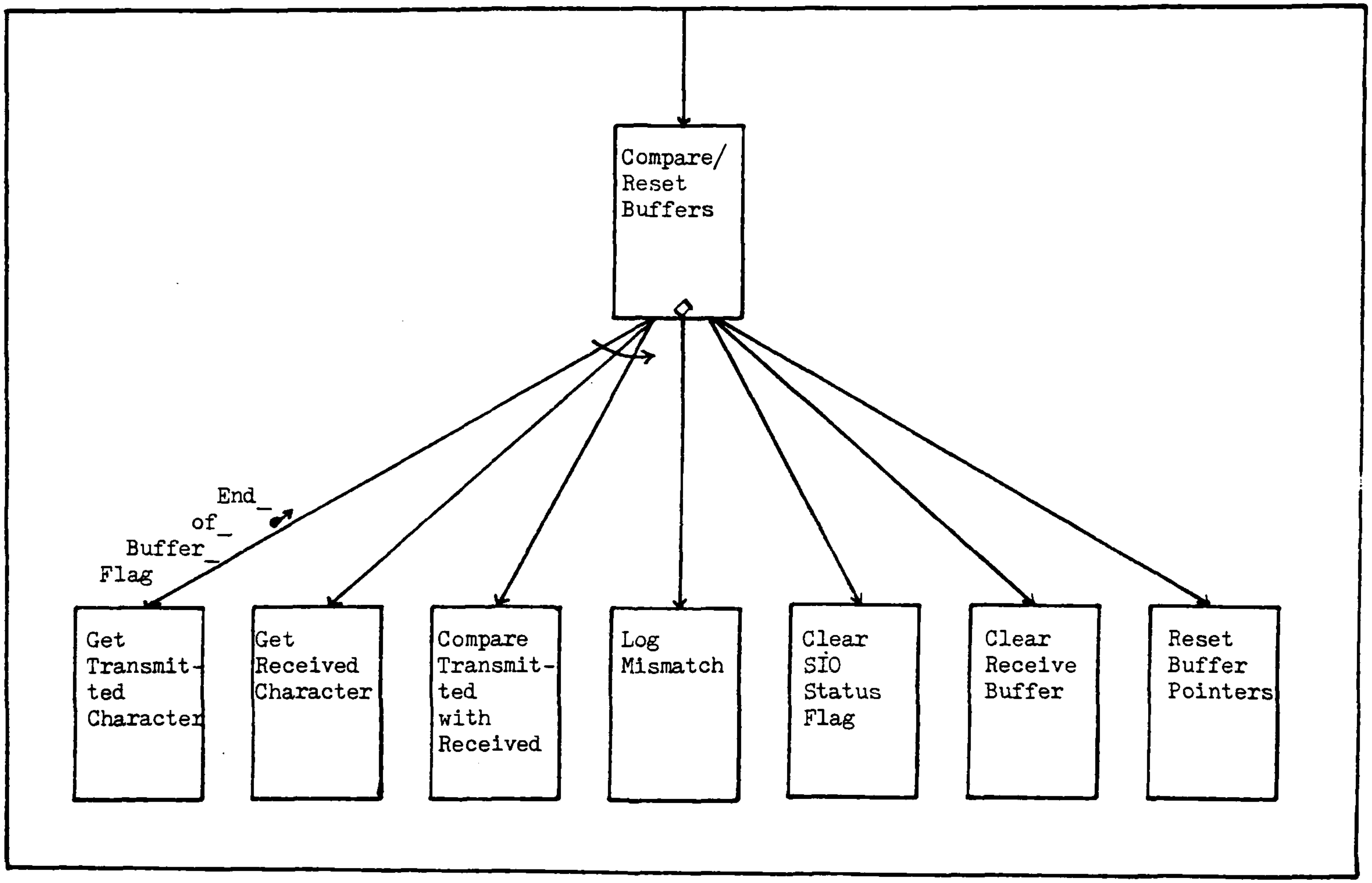
SIMULATE--Activate Transmit Mode on SIO Channel (Level 5)



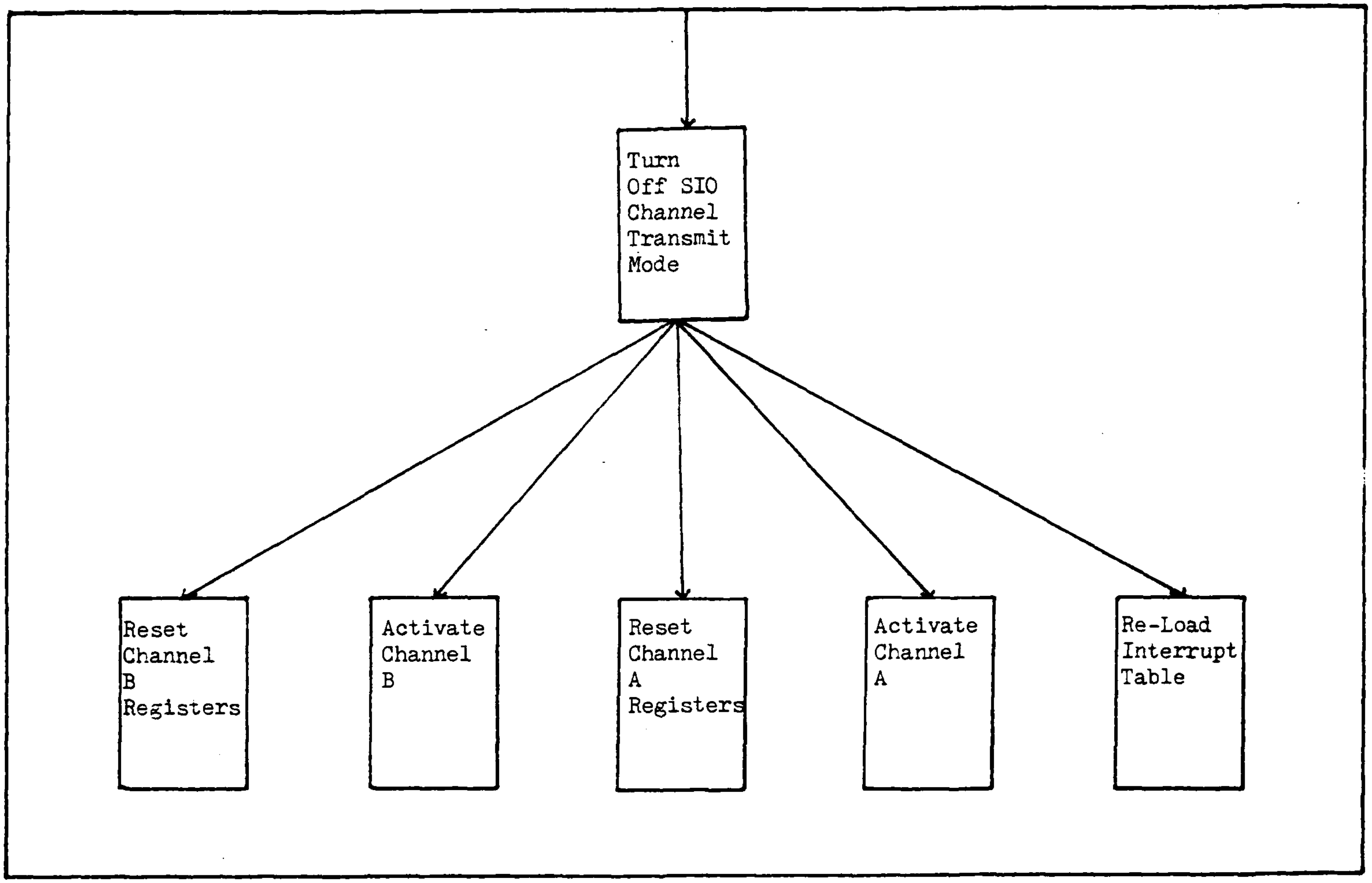
SIMULATE--Initialize Buffers and Counters Used in Simulate Tests (Level 5)



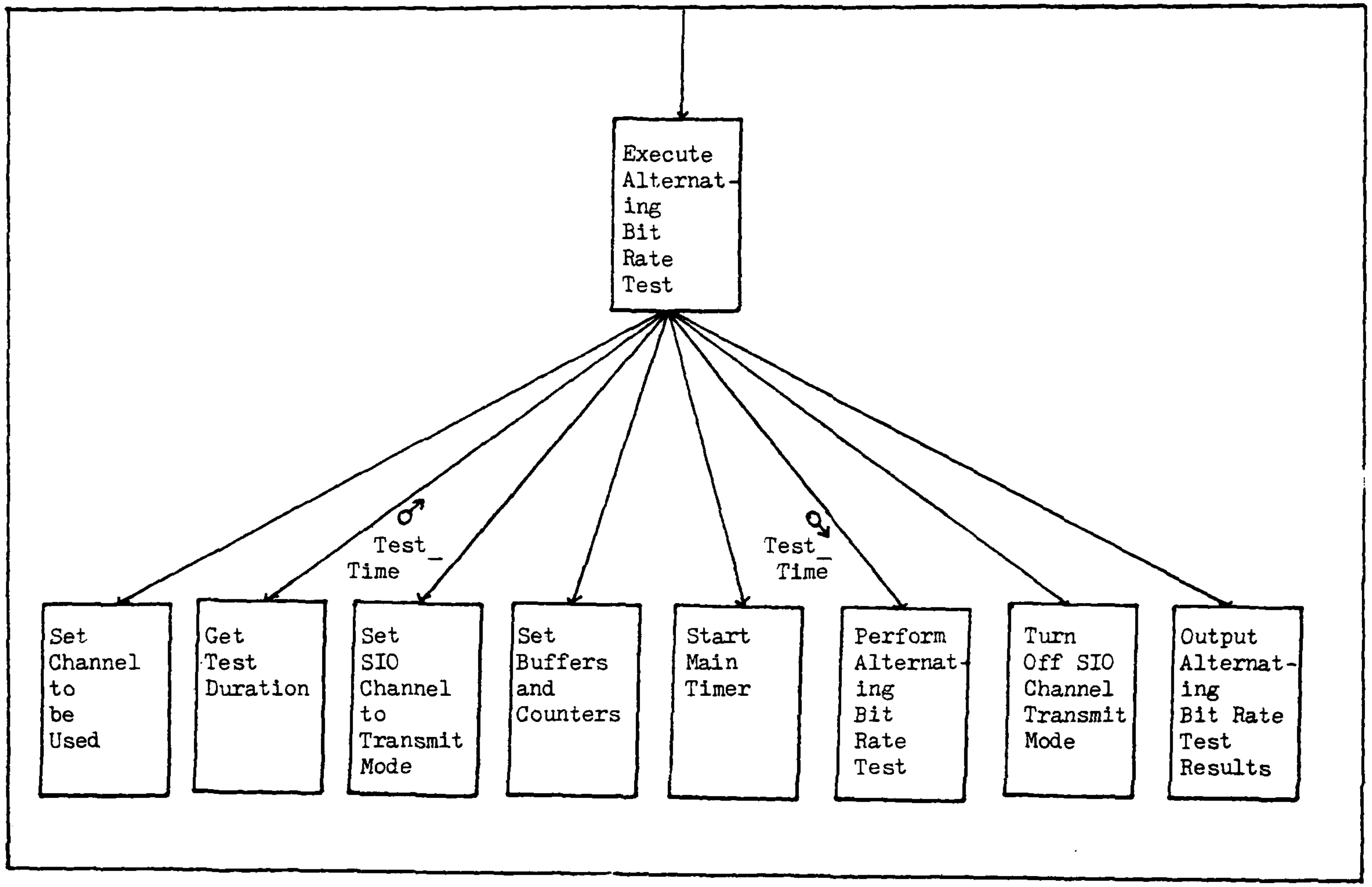
SIMULATE--Perform Bit Rate Test (Level 5)



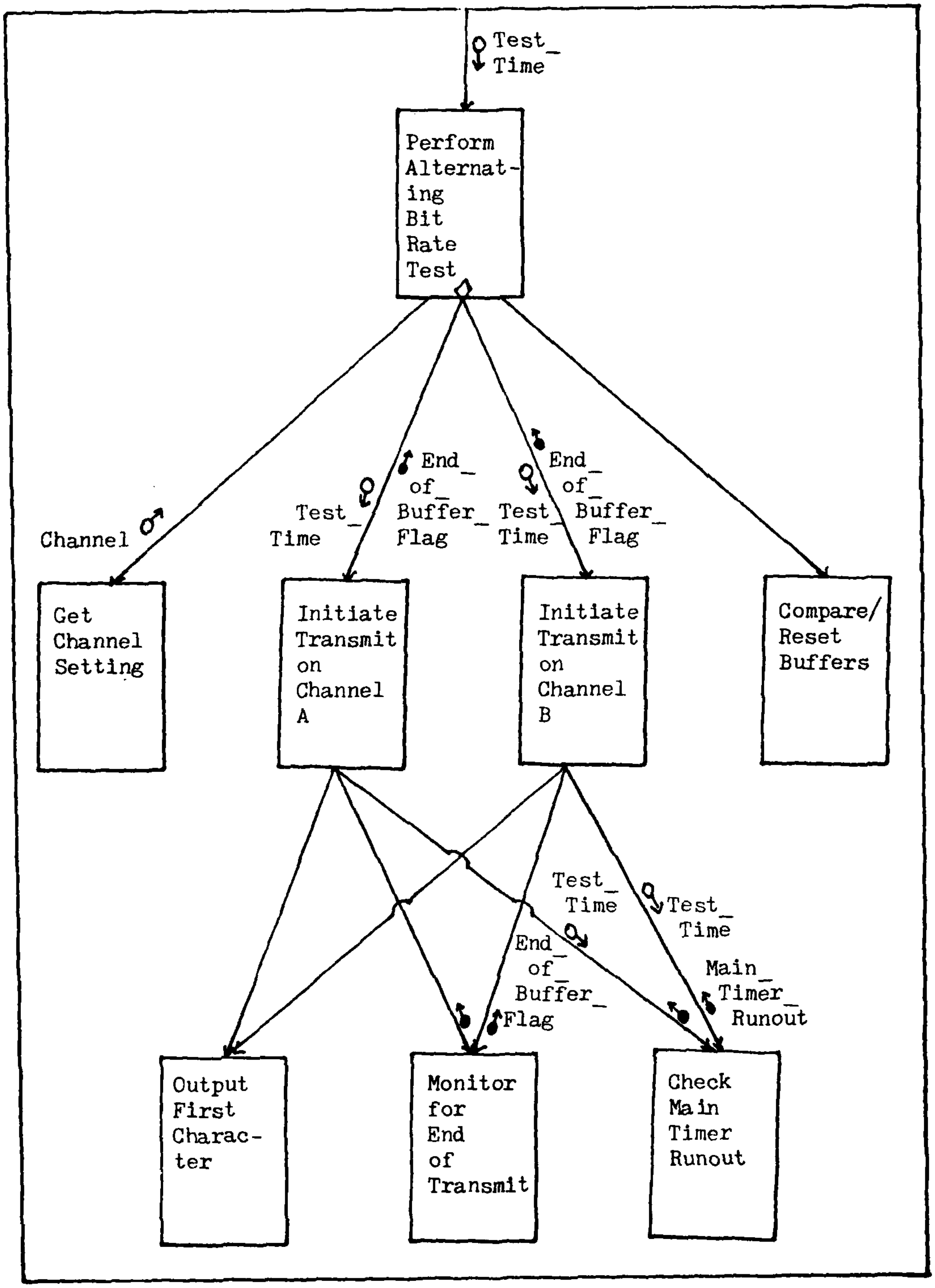
SIMULATE--Compare Transmitted Data with Received Data (Level 6)



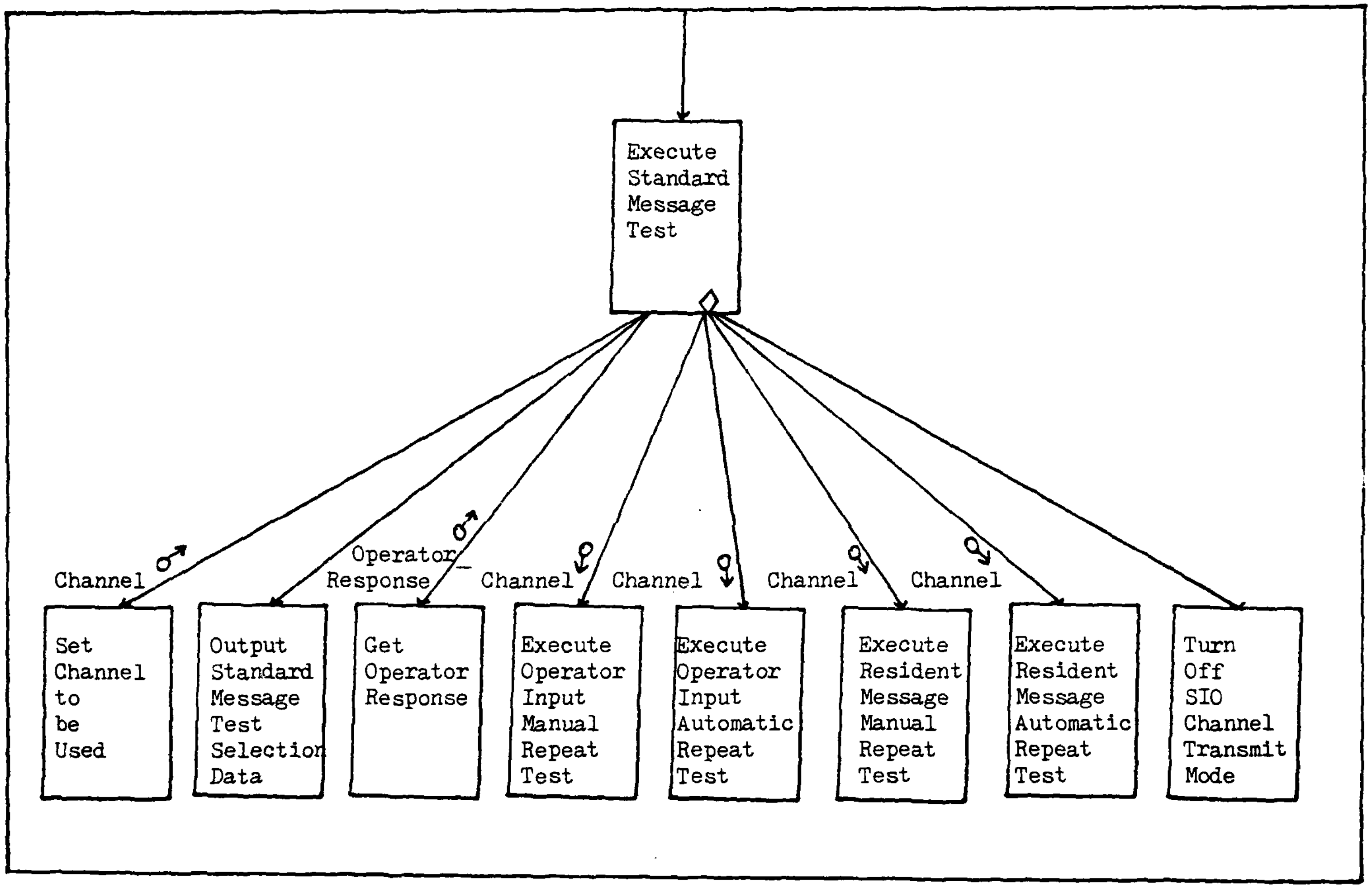
SIMULATE--De-activate the Transmit Mode on SIO Channel (Level 5)


Execute Alternating Bit Rate Test
Test_Time
Test_Time
Set Channel to be Used
Get Test Duration
Set SIO Channel to Transmit Mode
Set Buffers and Counters
Start Main Timer
Perform Alternating Bit Rate Test
Turn Off SIO Channel Transmit Mode
Output Alternating Bit Rate Test Results

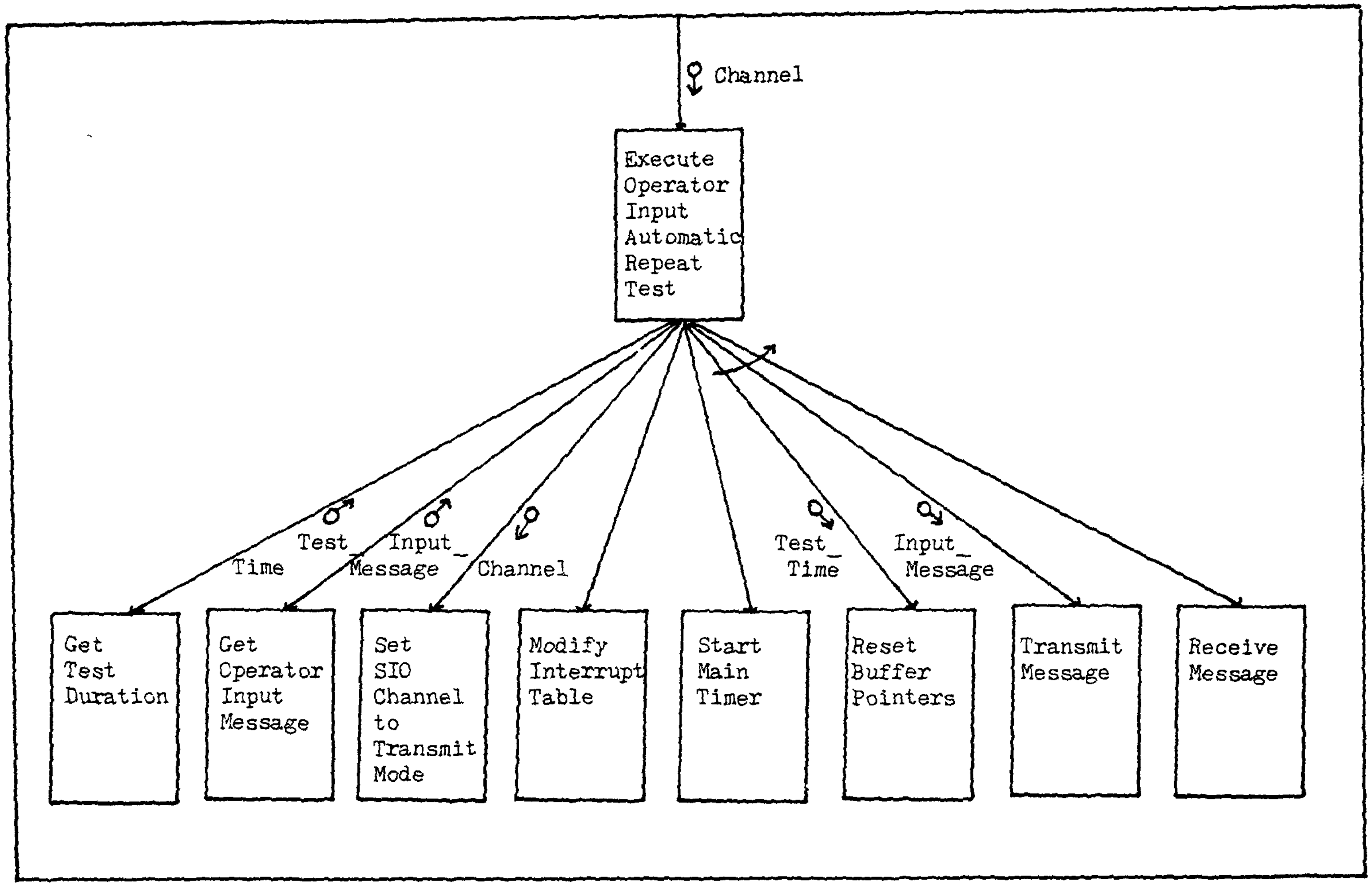
SIMULATE--Execute Alternating Bit Rate Test (Level 4)



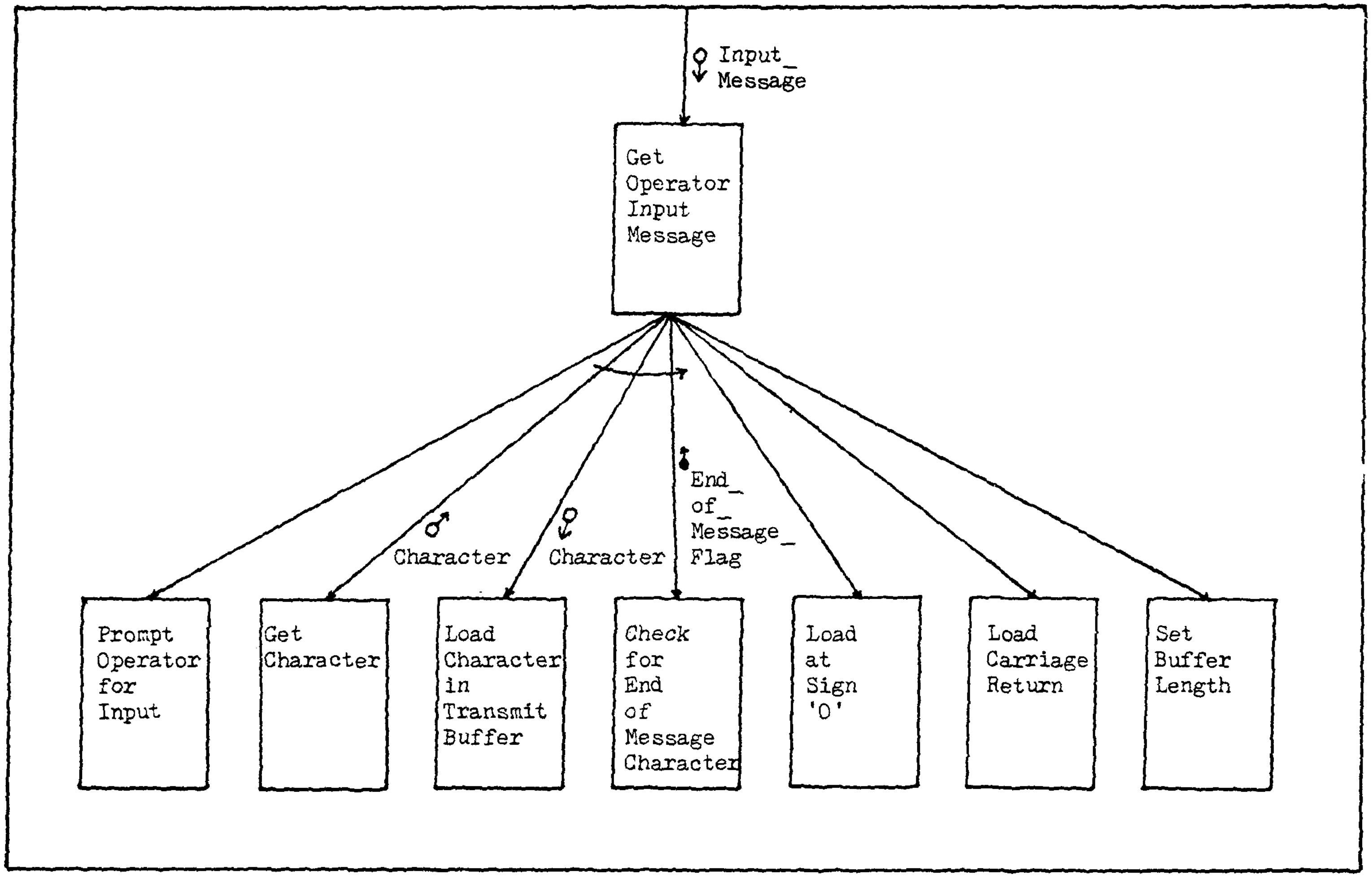
SIMULATE--Perform Alternating Bit Rate Test (Level 5)


Execute Standard Message Test
Channel
Operator Response
Channel
Channel
Channel
Channel
Set Channel to be Used
Output Standard Message Test Selection Data
Get Operator Response
Execute Operator Input Manual Repeat Test
Execute Operator Input Automatic Repeat Test
Execute Resident Message Manual Repeat Test
Execute Resident Message Automatic Repeat Test
Turn Off SIO Channel Transmit Mode

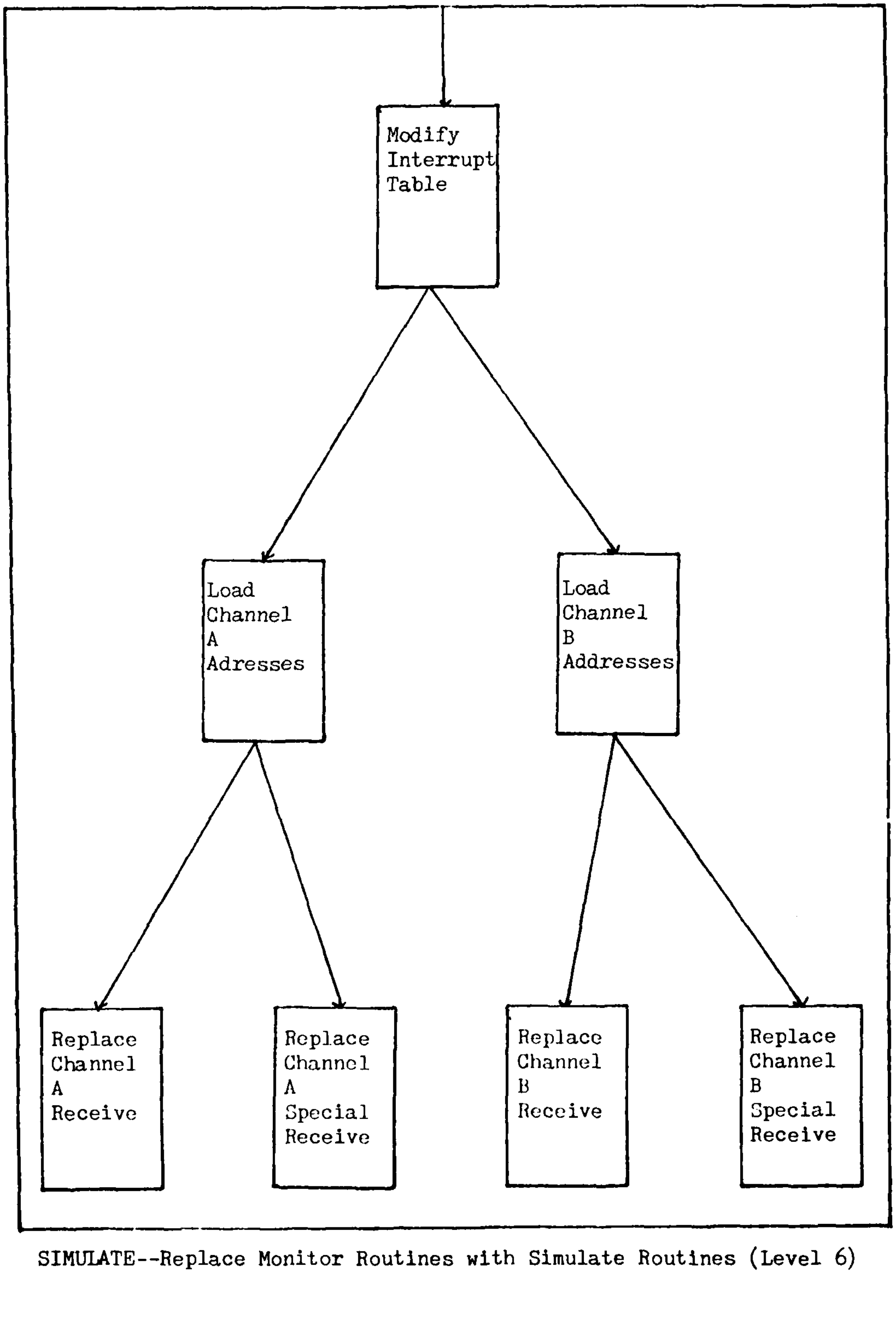
SIMULATE--Execute Standard Message Test (Level 4)



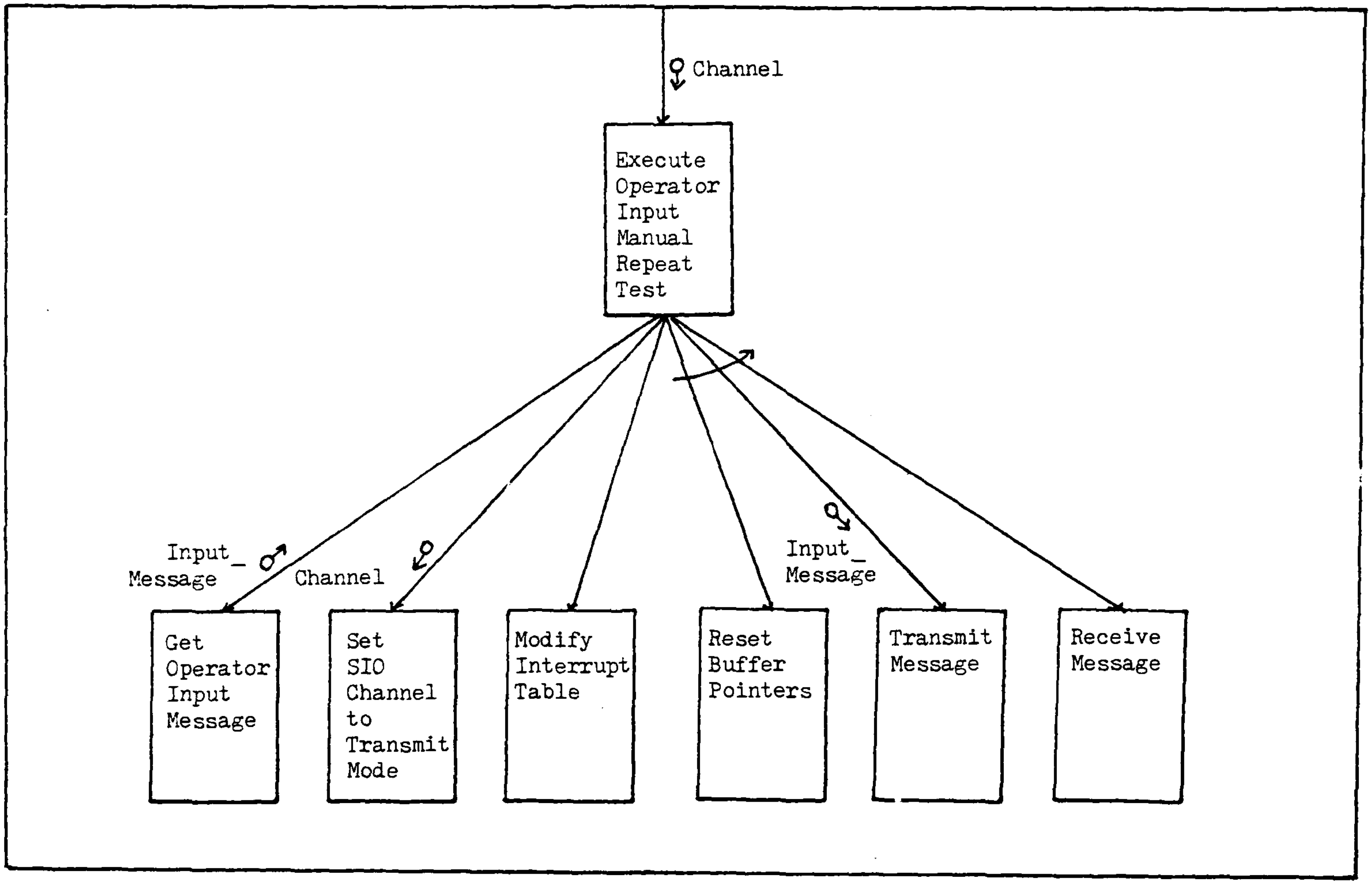
SIMULATE--Execute Operator Input Automatic Repeat Test (Level 5)



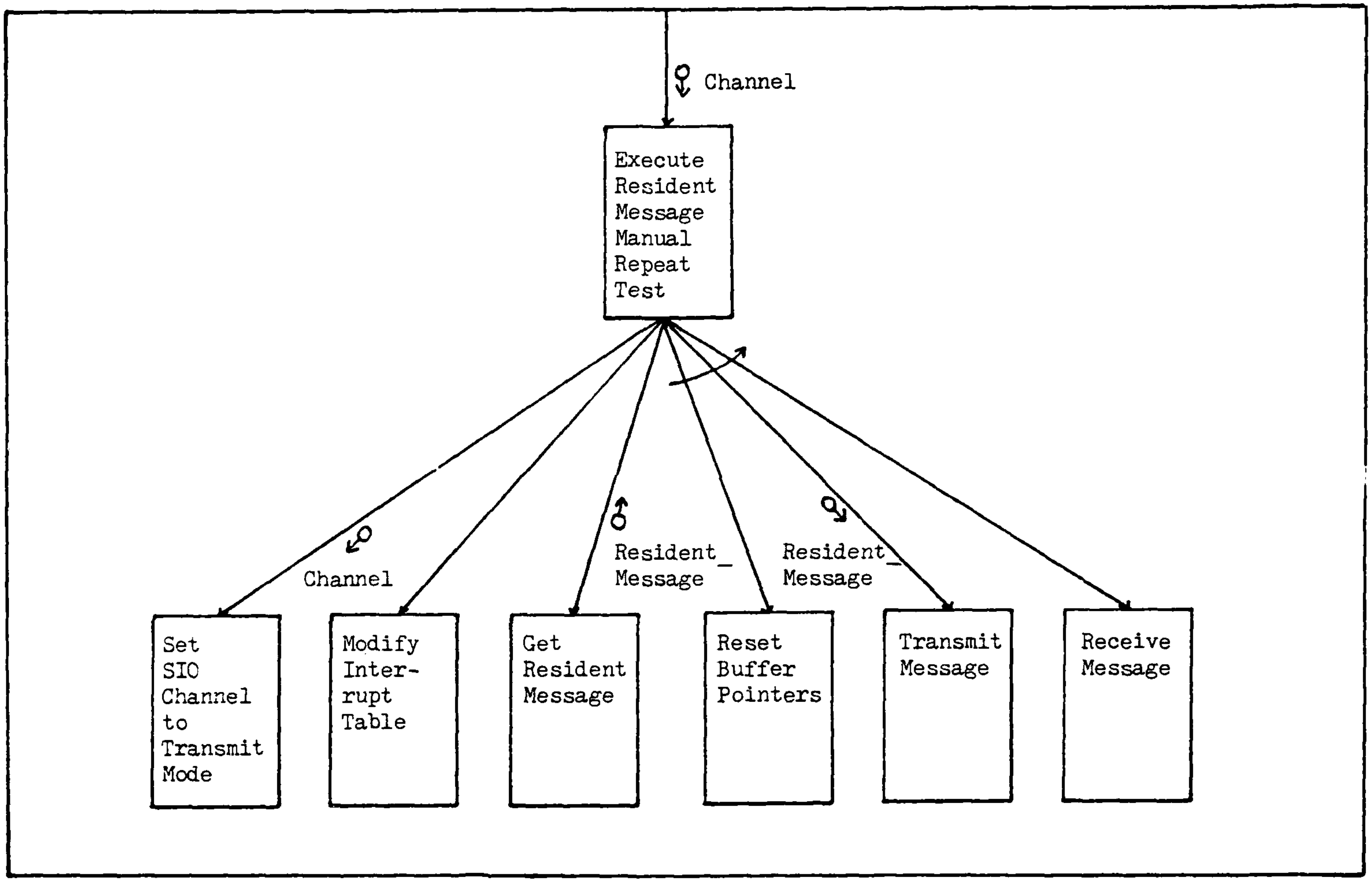
SIMULATE--Get Message from Operator to be Transmitted (Level 6)



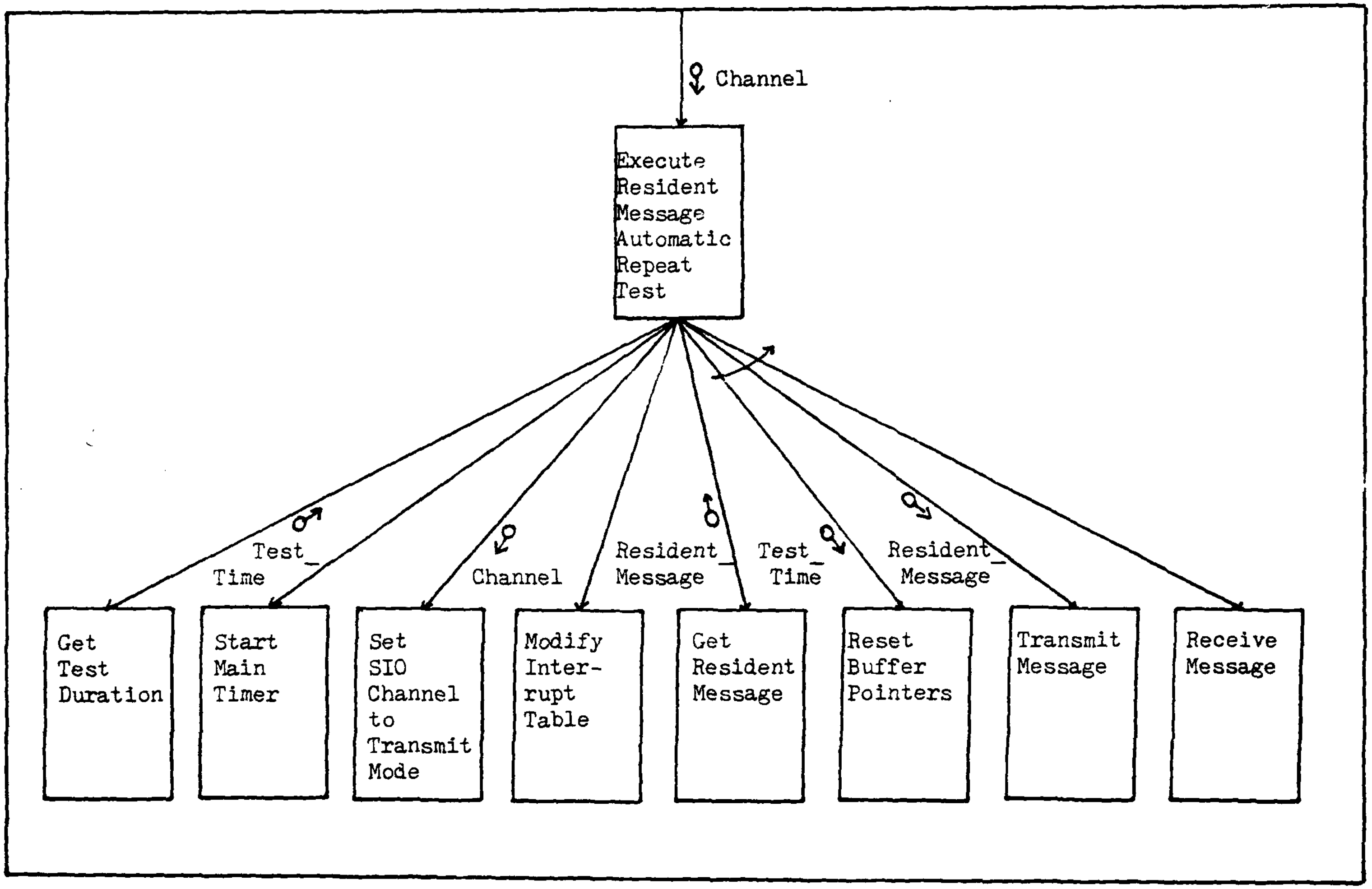
SIMULATE--Replace Monitor Routines with Simulate Routines (Level 6)



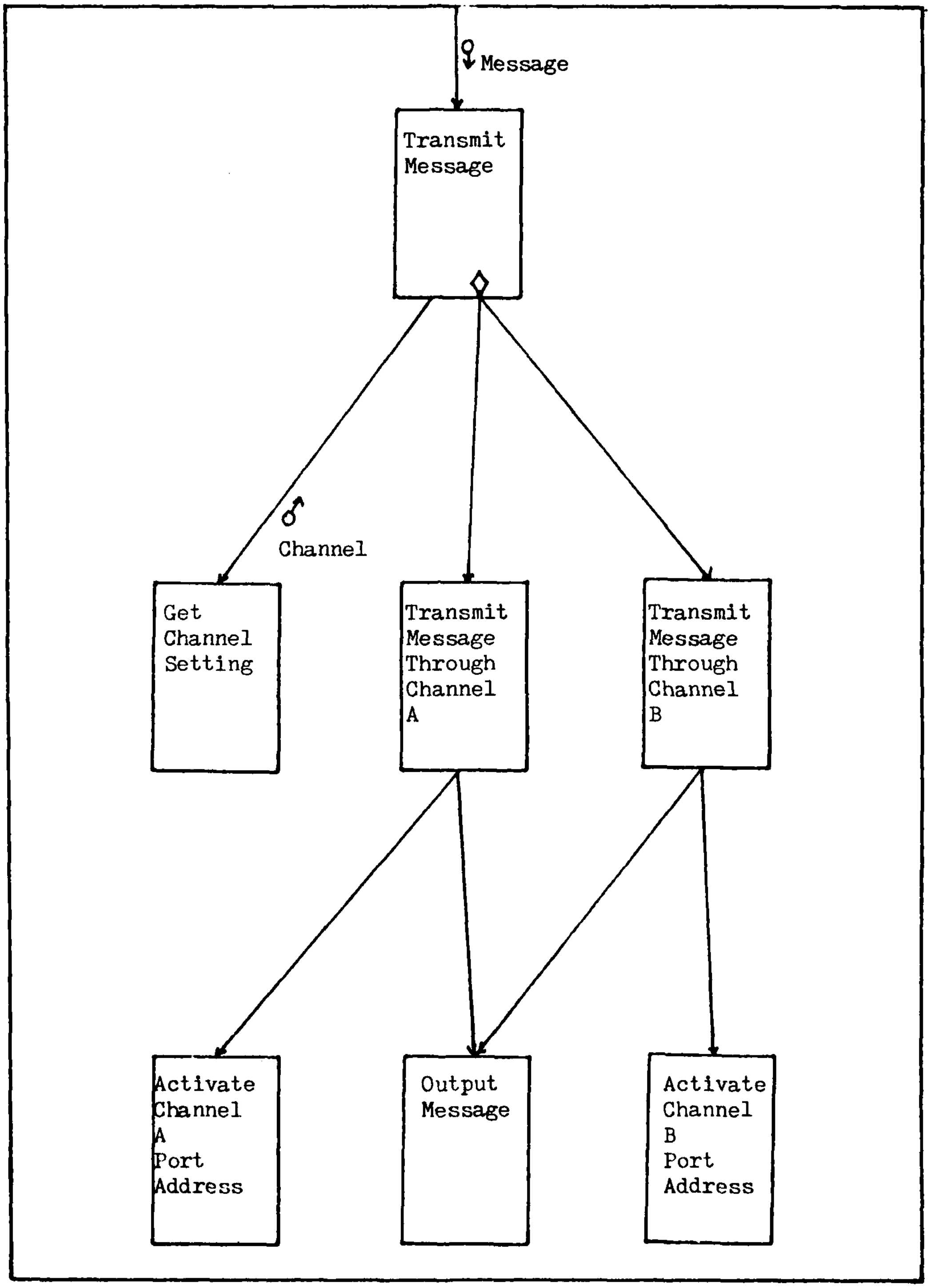
SIMULATE--Execute Operator Input Manual Repeat Test (Level 5)



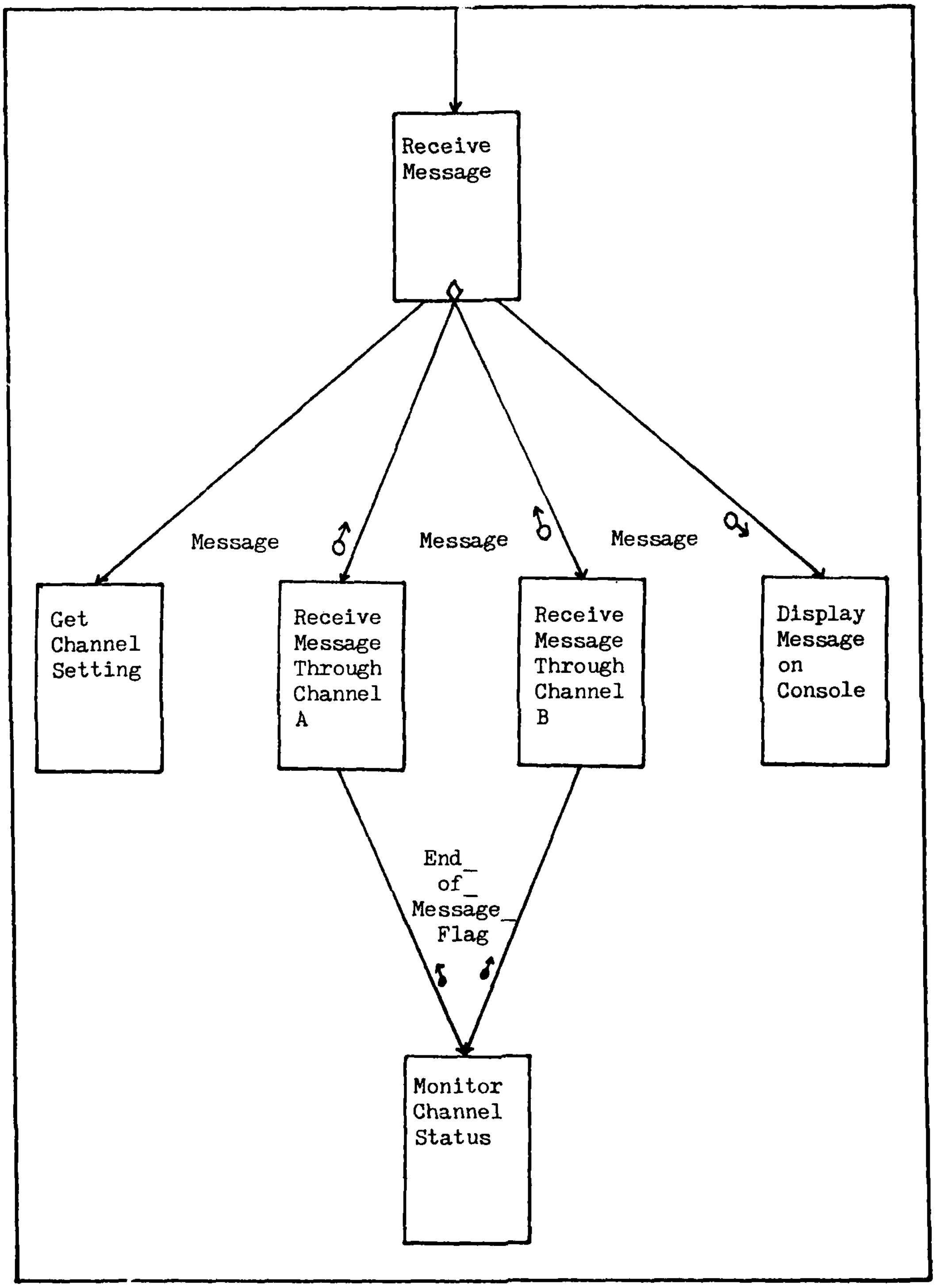
SIMULATE--Execute Resident Message Manual Repeat Test (Level 5)



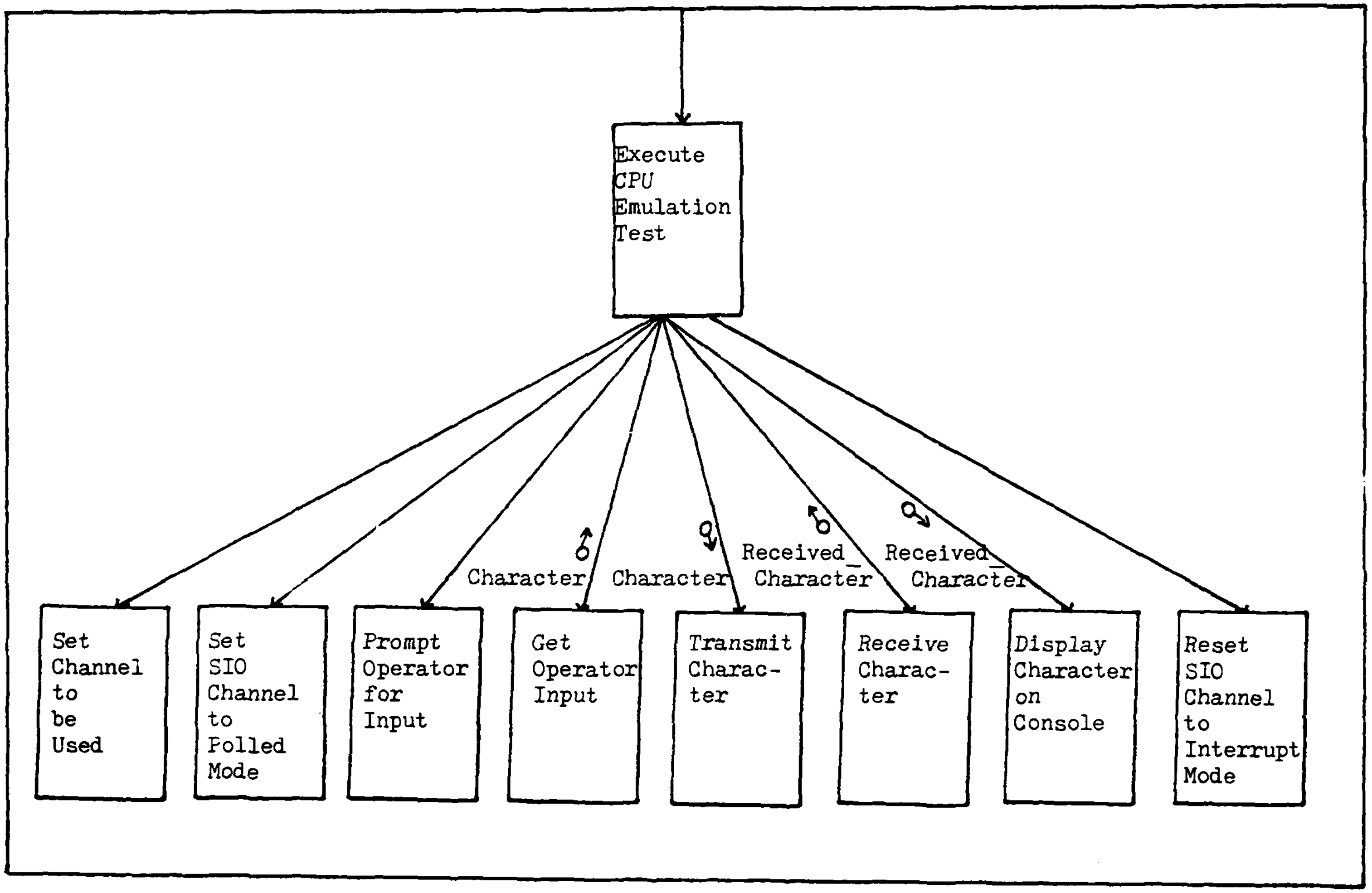
SIMULATE--Execute Resident Message Automatic Repeat Test (Level 5)


Message
Transmit Message
Channel
Get Channel Setting
Transmit Message Through Channel A
Transmit Message Through Channel B
Activate Channel A Port Address
Output Message
Activate Channel B Port Address

SIMULATE--Transmit Standard Message (Level 6)

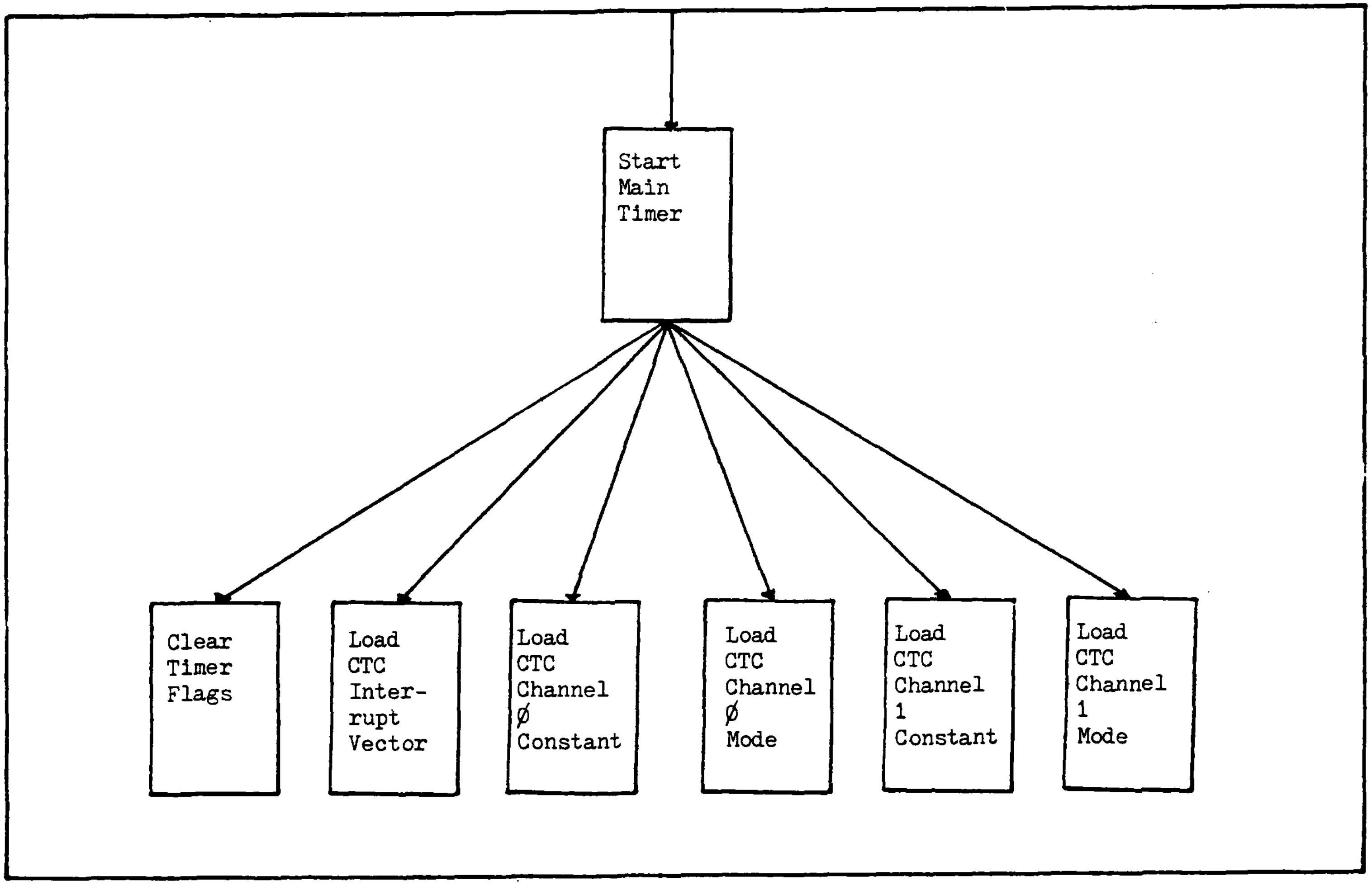


SIMULATE--Receive Standard Message (Level 6)

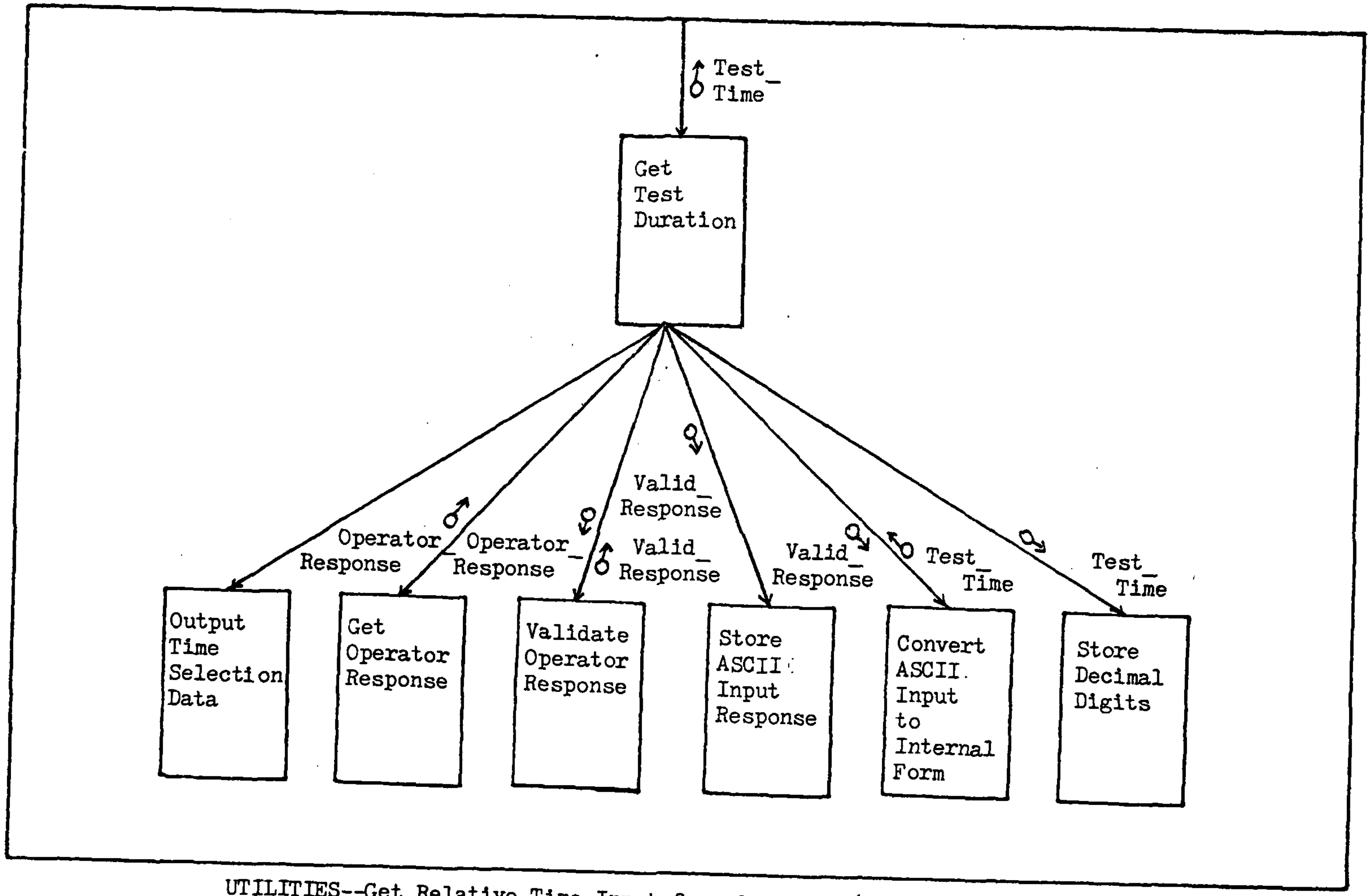


Execute CPU Emulation Test
Character
Character
Received_Character
Received_Character
Set Channel to be Used
Set SIO Channel to Polled Mode
Prompt Operator for Input
Get Operator Input
Transmit Charac-ter
Receive Charac-ter
Display Character on Console
Reset SIO Channel to Interrupt Mode

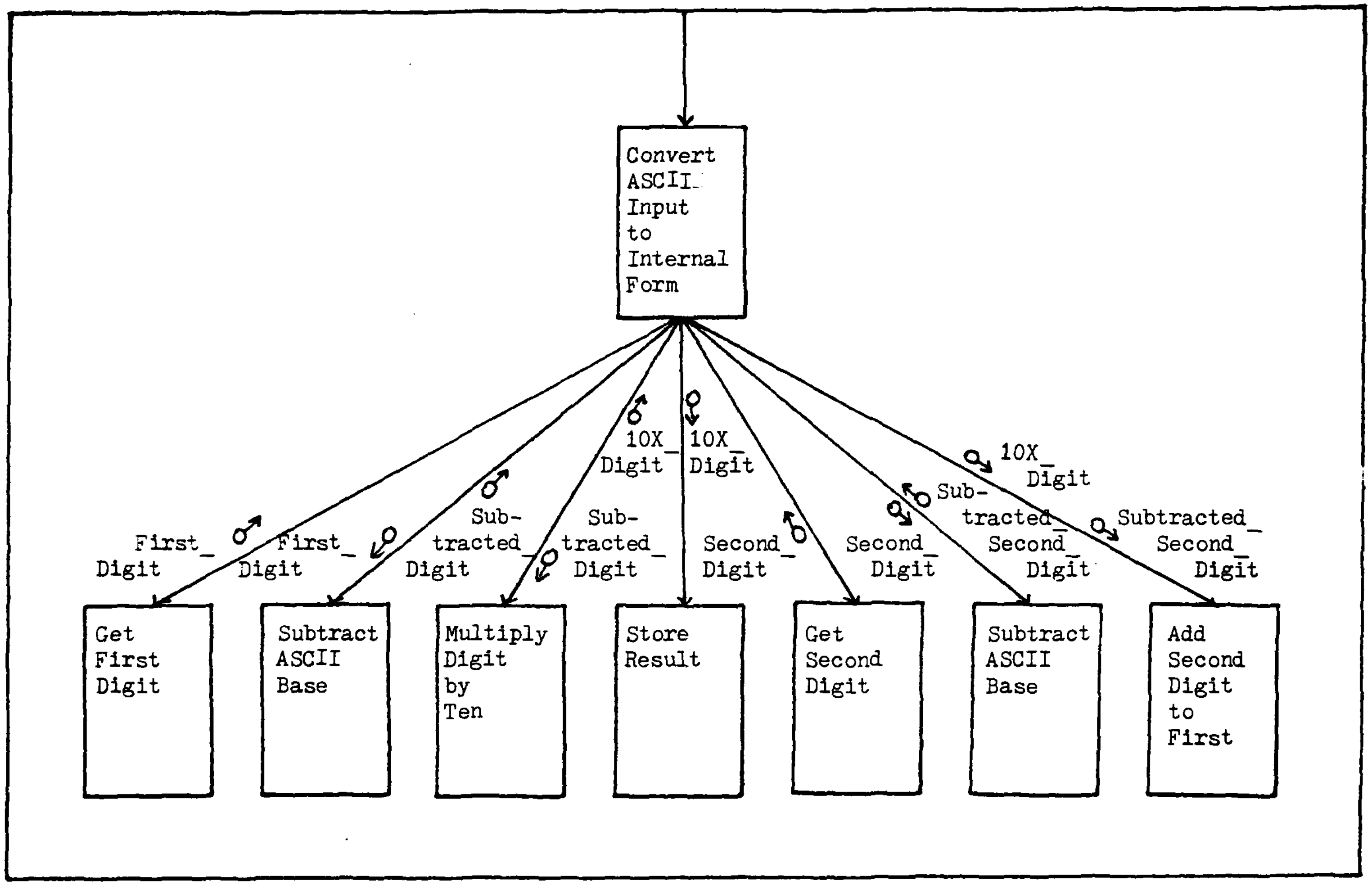
SIMULATE--Execute CPU Emulation Test (Level 4)



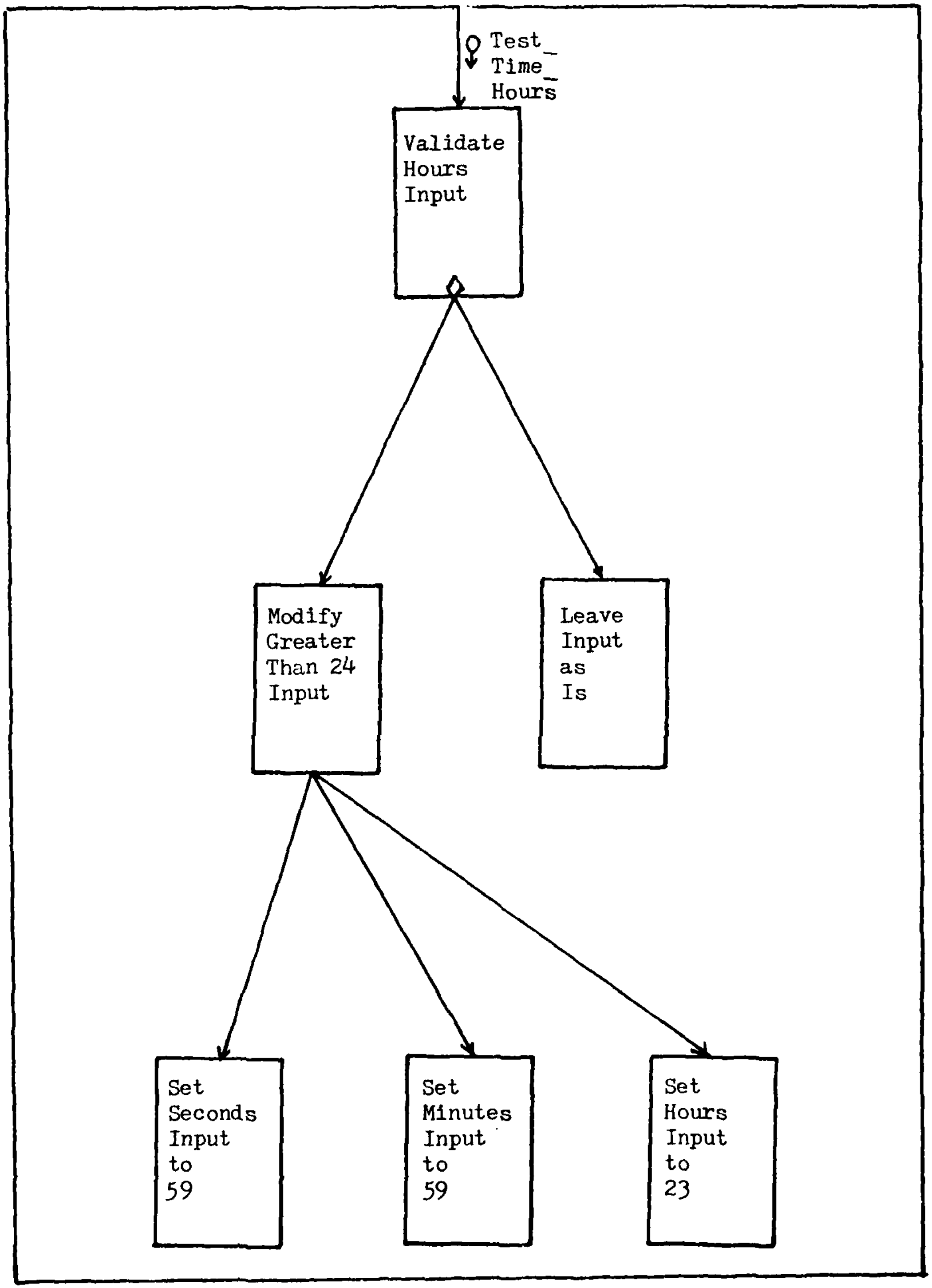
UTILITIES--Activate the Main Timer



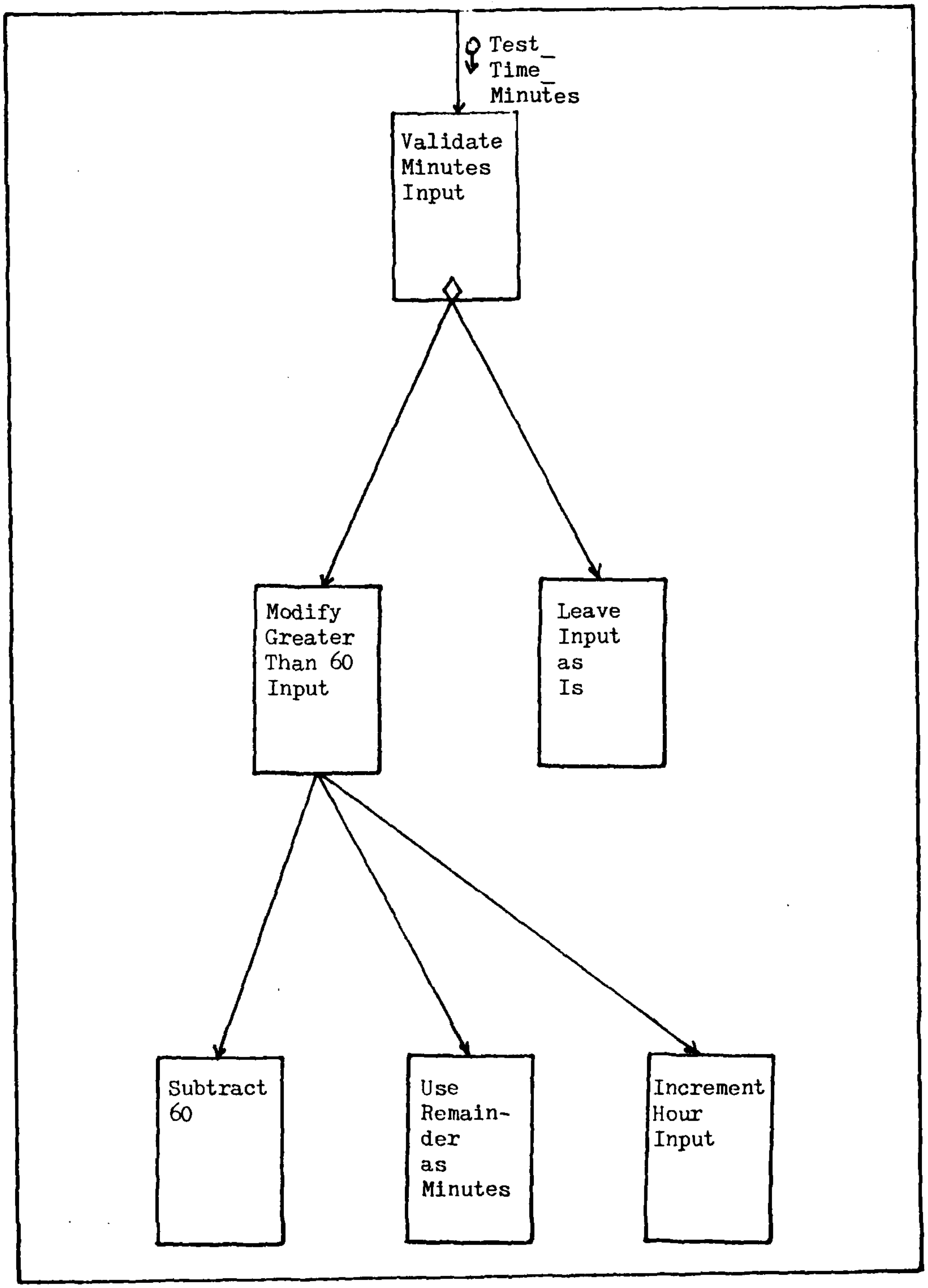
UTILITIES--Get Relative Time Input from Operator (Level 5)



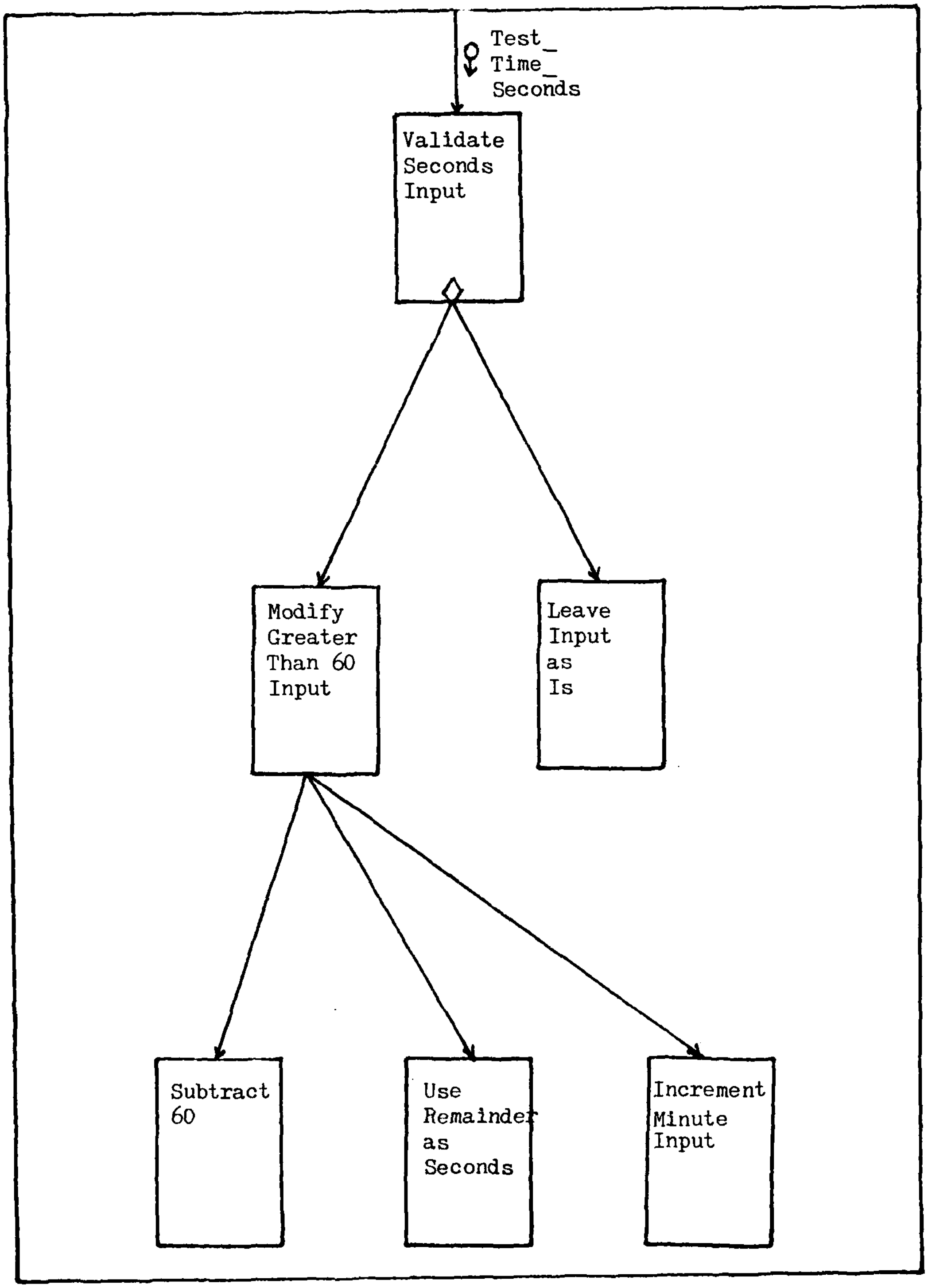
UTILITIES--Convert ASCII Input of Operator into Internal Form (Level 6)



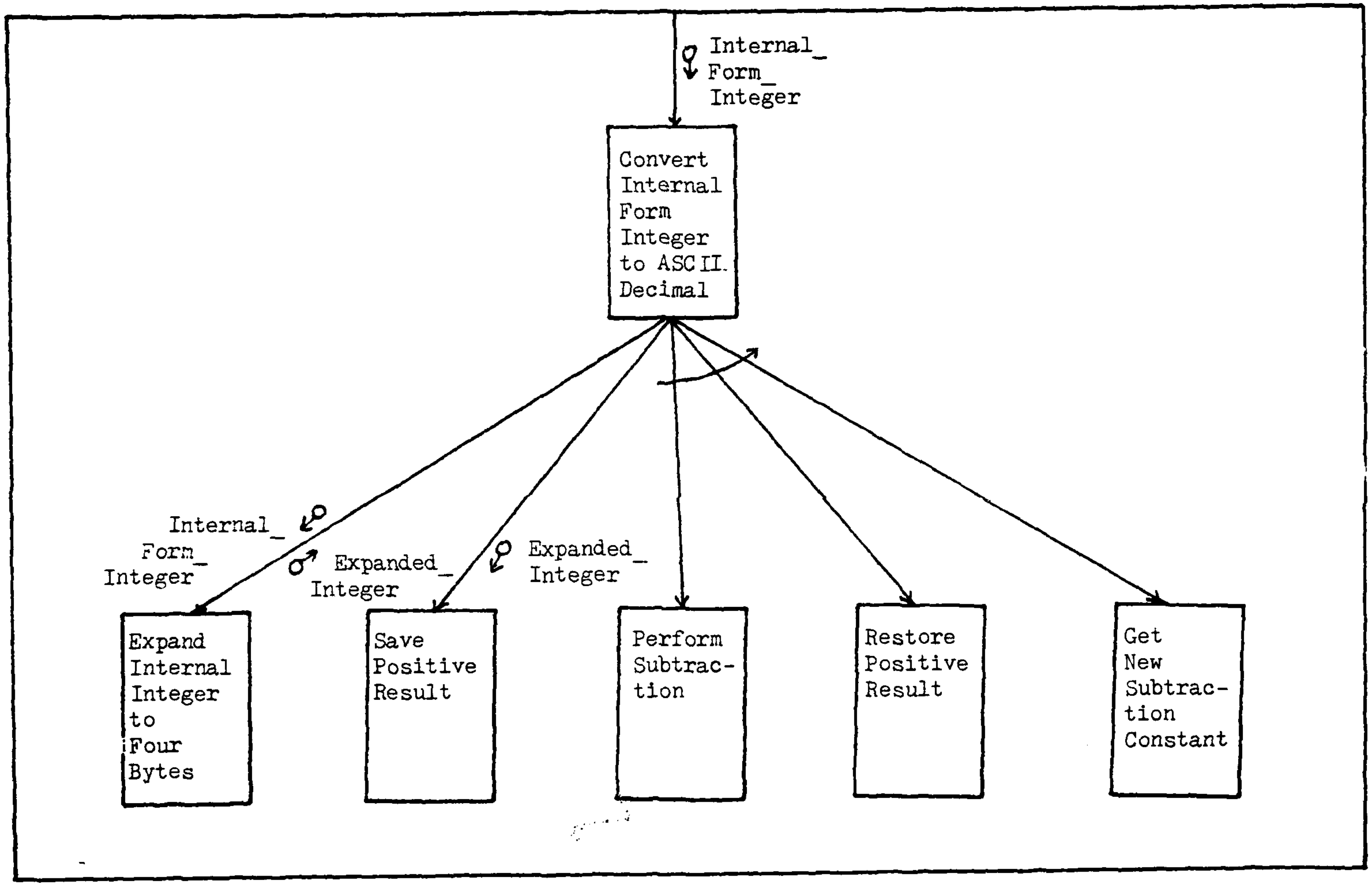
UTILITIES--Validate Operator Response-Time(Hours) Input (Level 6)



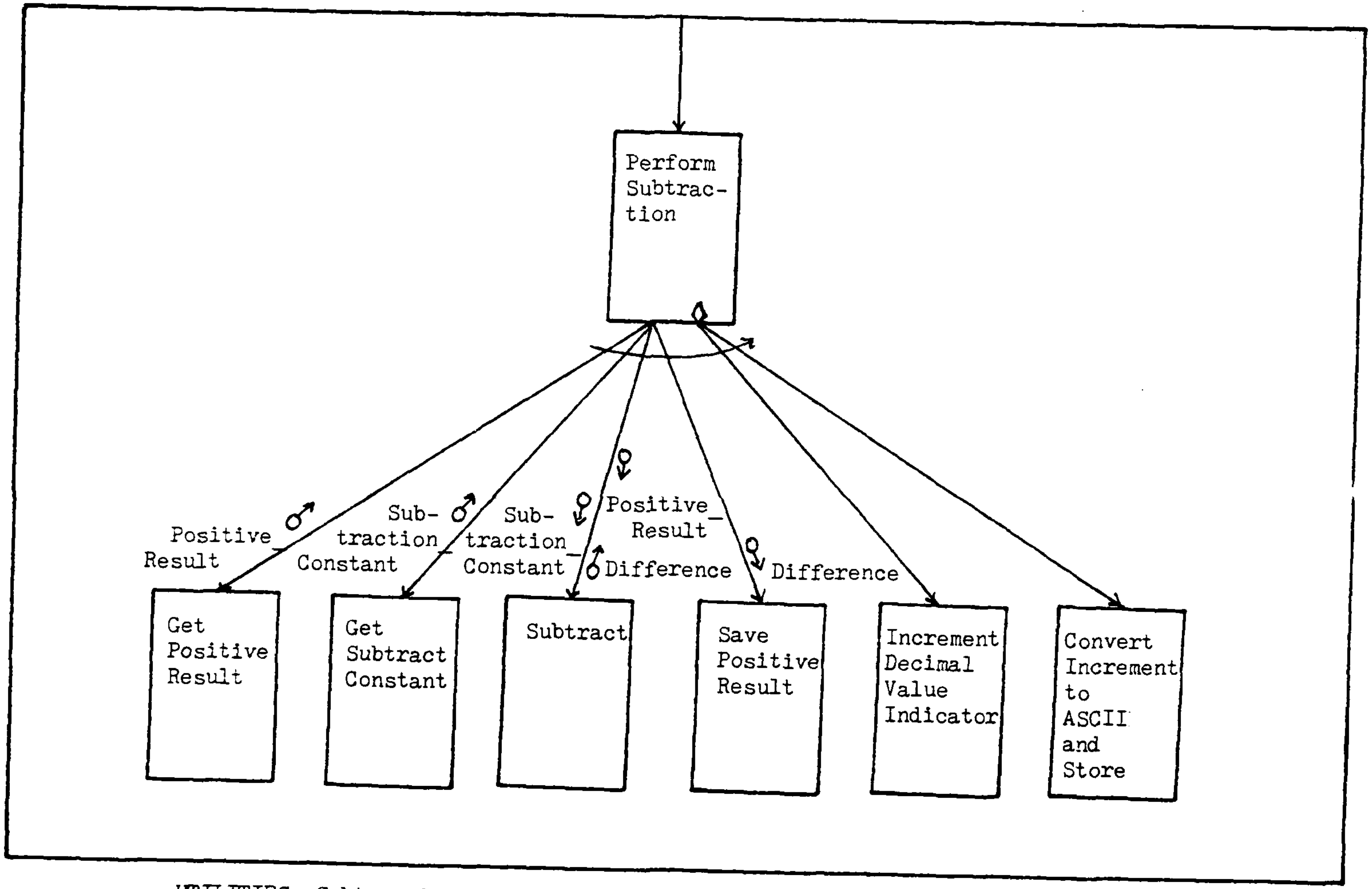
UTILITIES--Validate Operator Input-Time(Minutes) Input (Level 6)



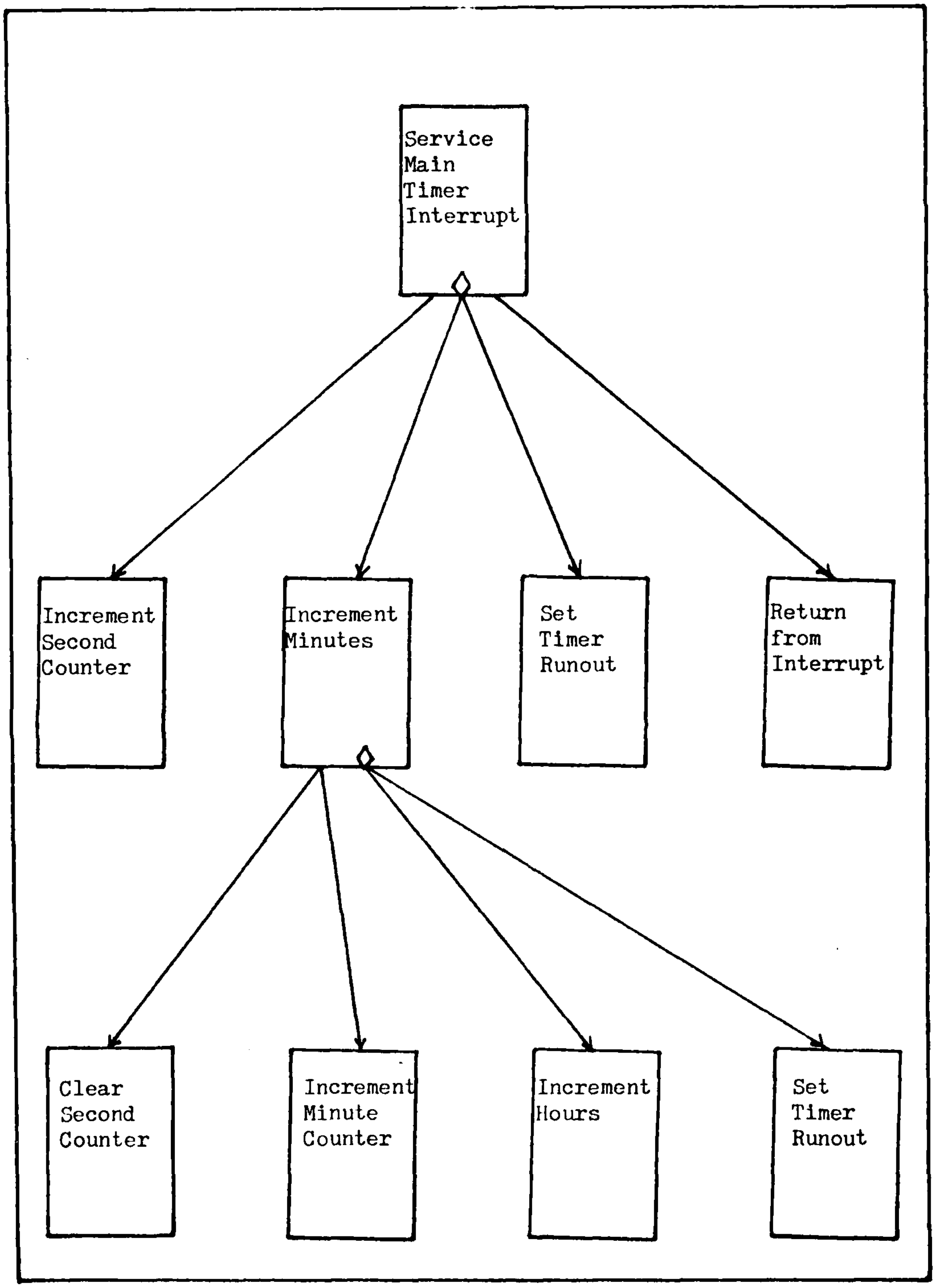
UTILITIES--Validate Operator Input-Time(Seconds) Input (Level 6)



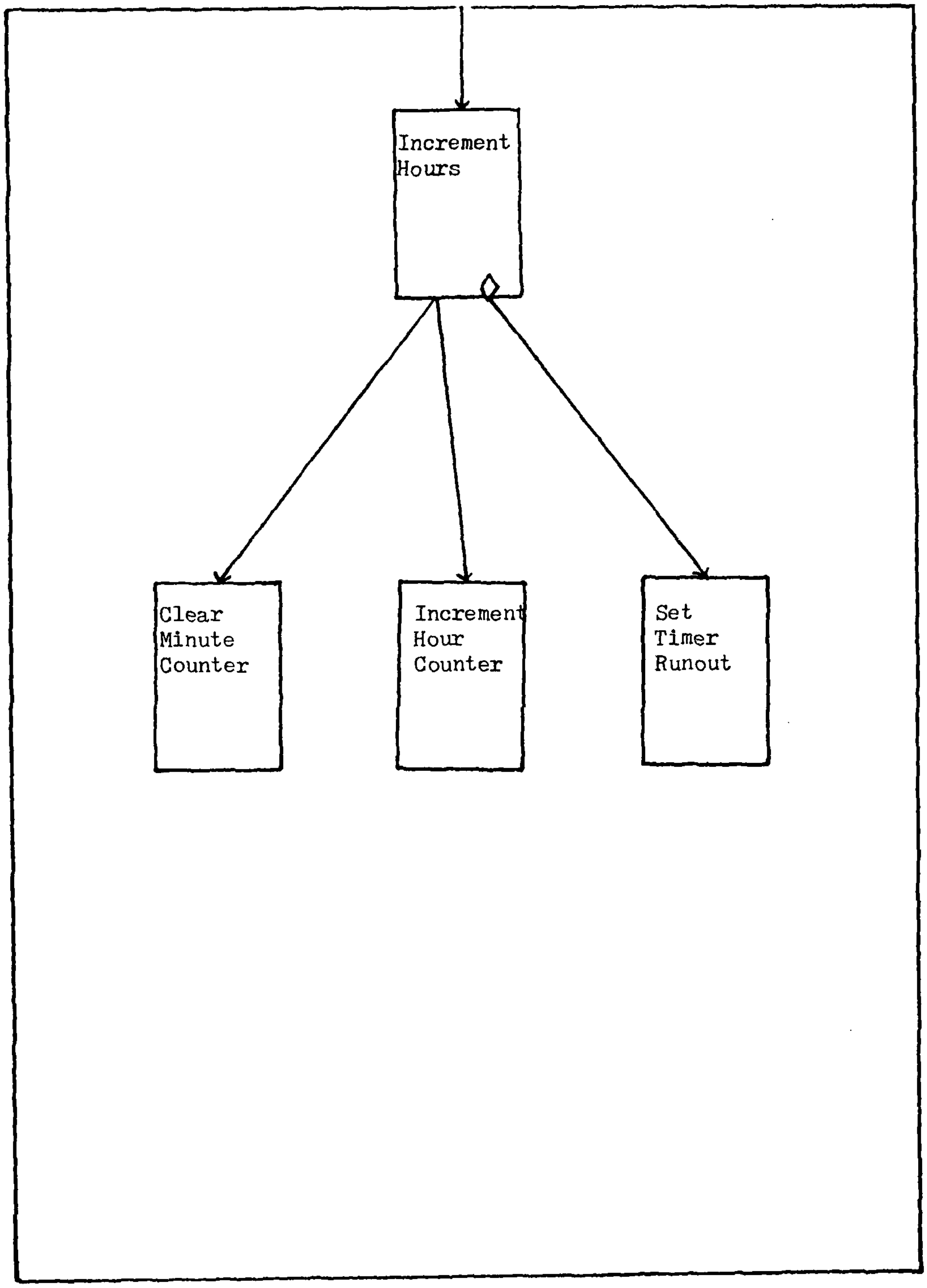
UTILITIES--Convert Internal Form Integer to ASCII Decimal.(Level 6)



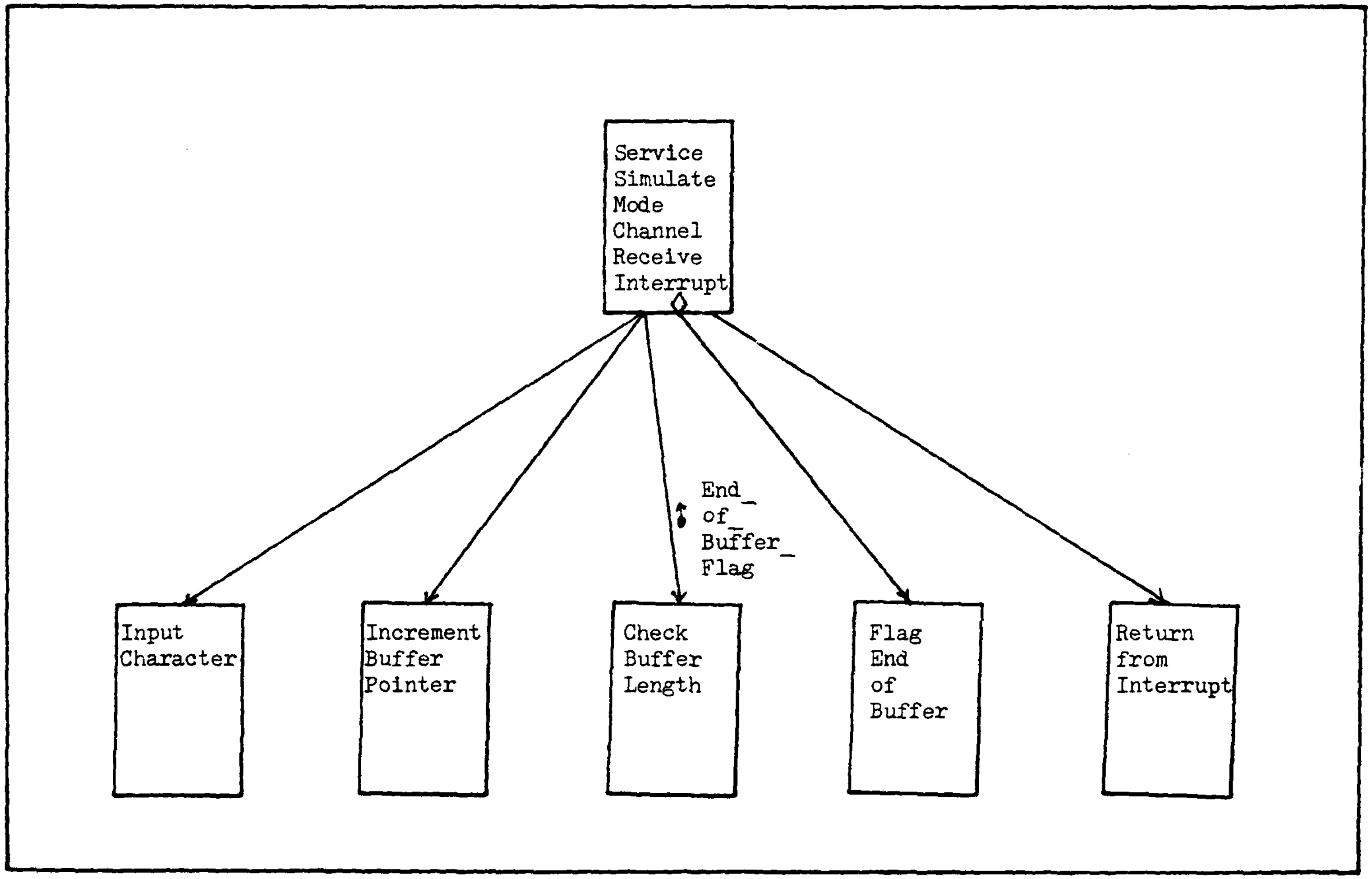
UTILITIES--Subtraction Routine to Convert Internal Form to ASCII (Level 7)



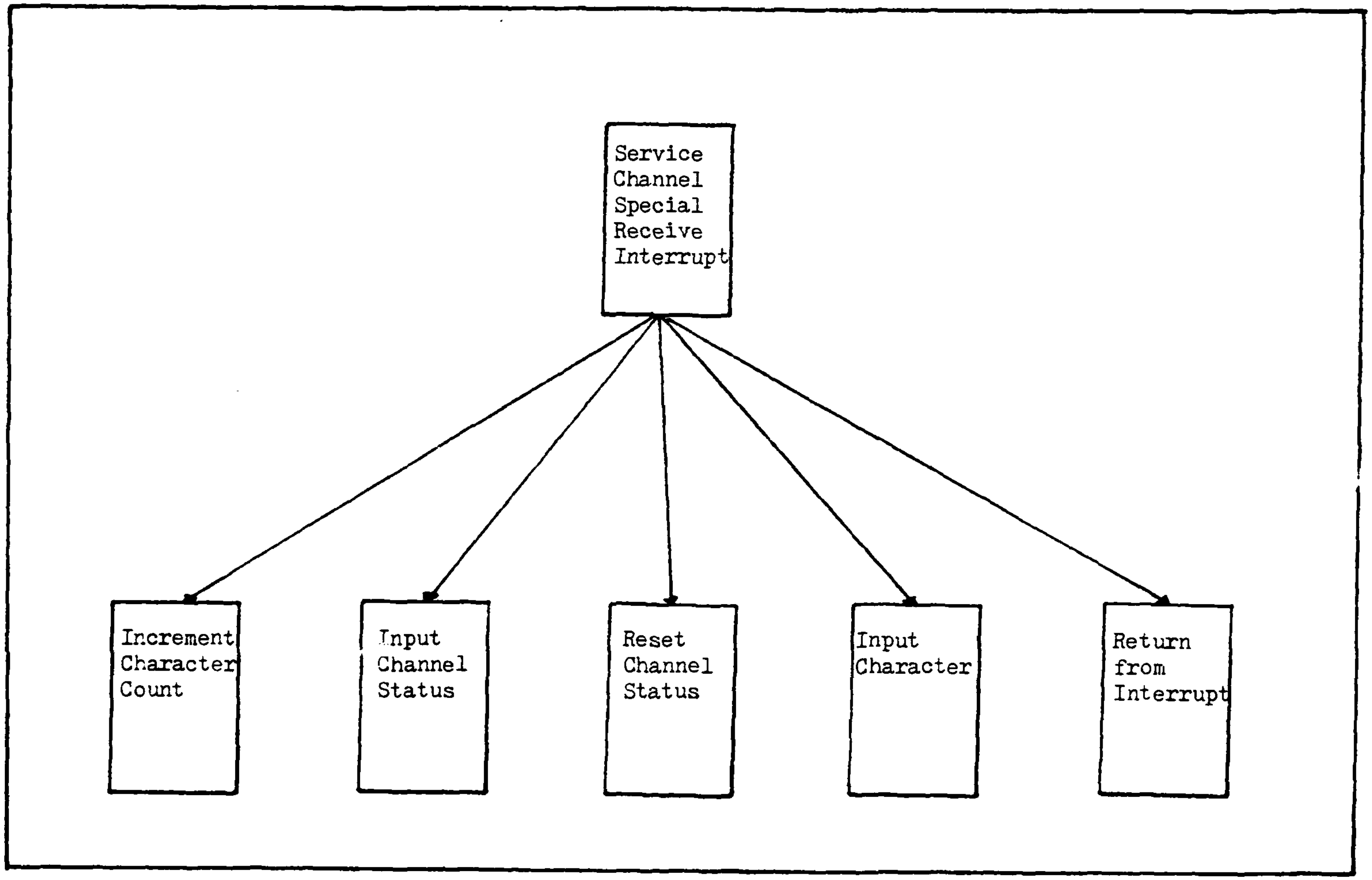
UTILITIES--Main Timer Interrupt Service Routine (Level 1)



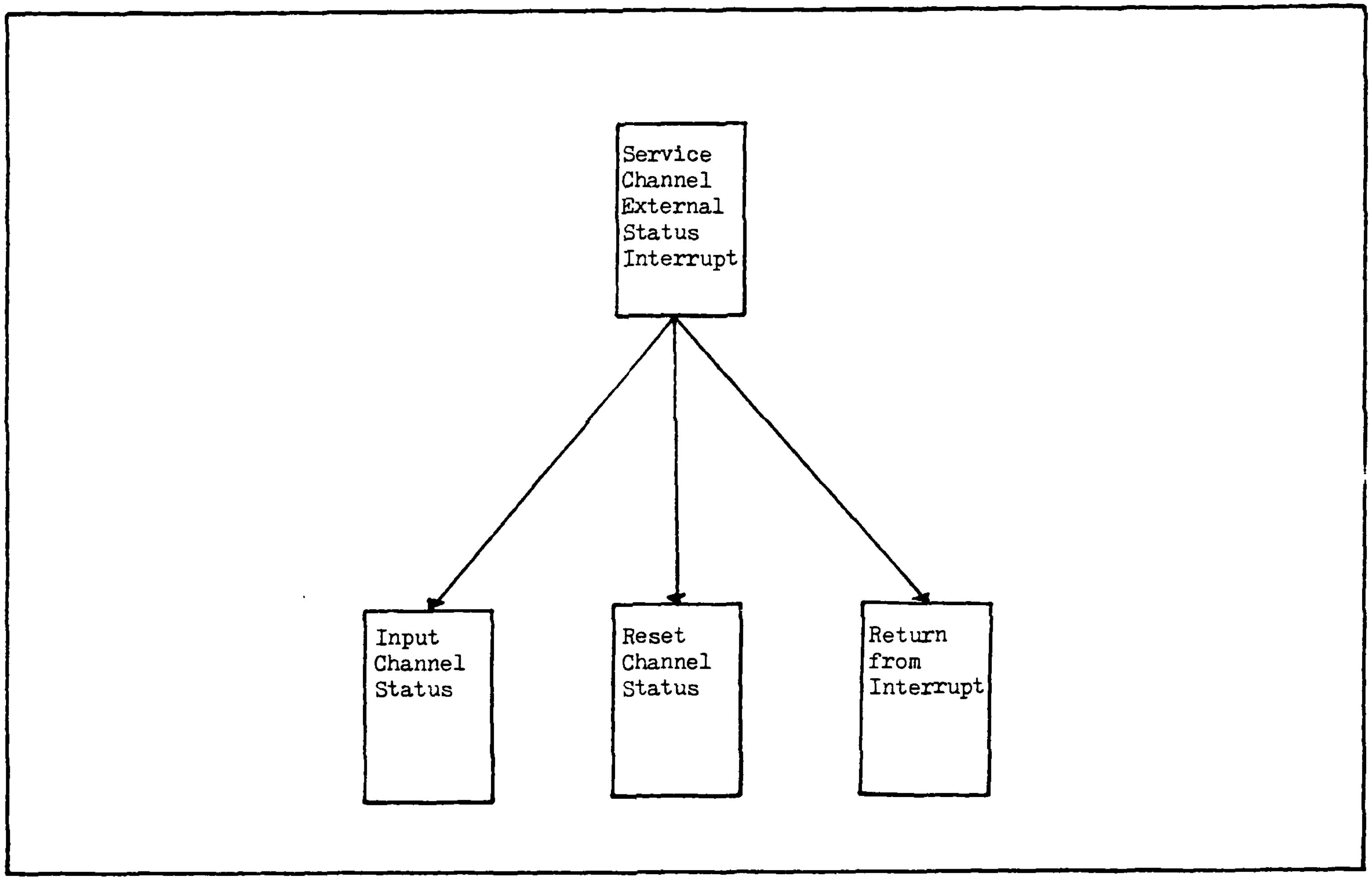
UTILITIES--Main Timer Interrupt Service Routine (Level 2)



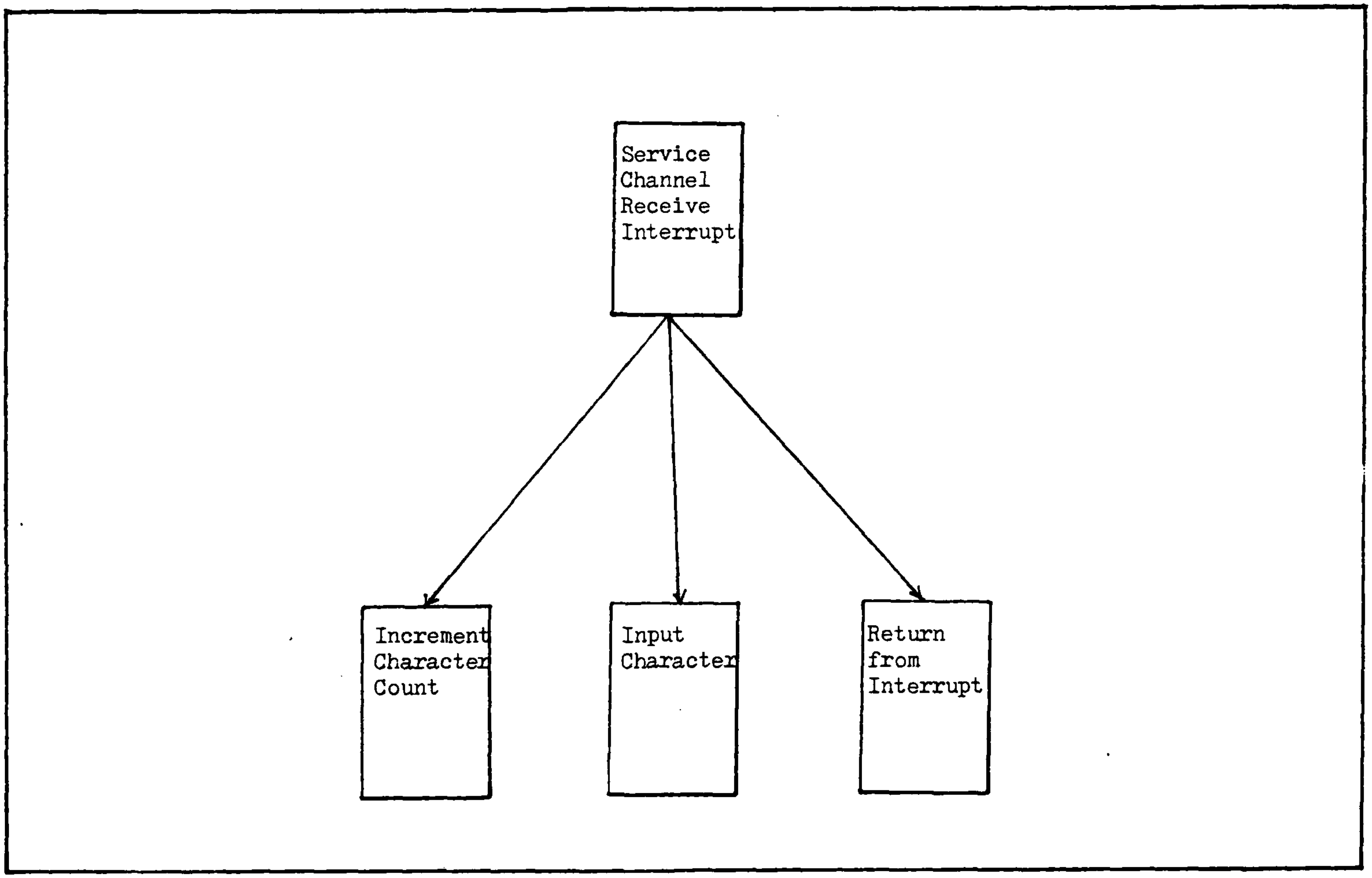
UTILITIES--Simulate Mode Channel Receive Interrupt Service Routine



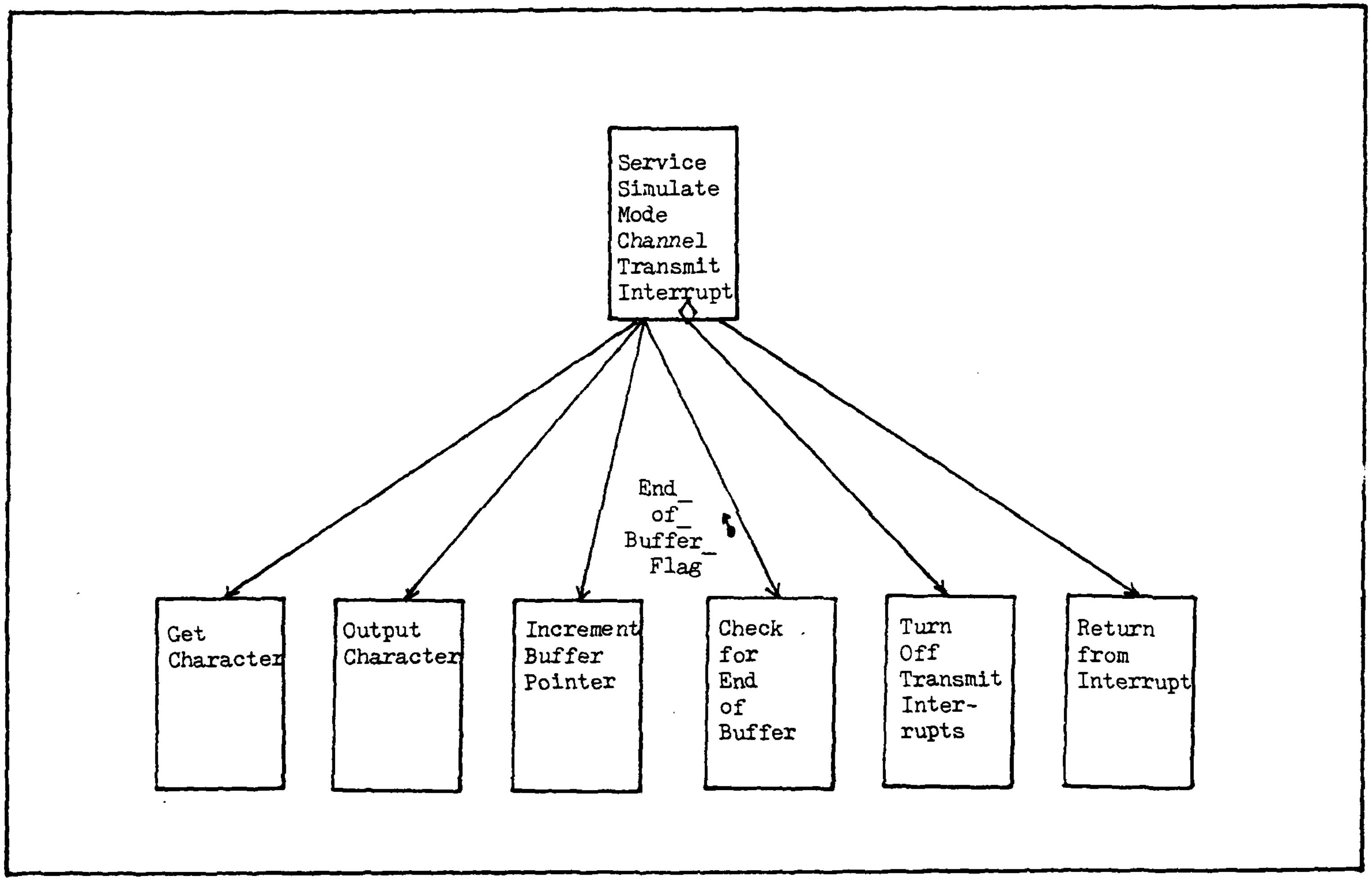
UTILITIES--Monitor Mode Channel Special Receive Interrupt Routine



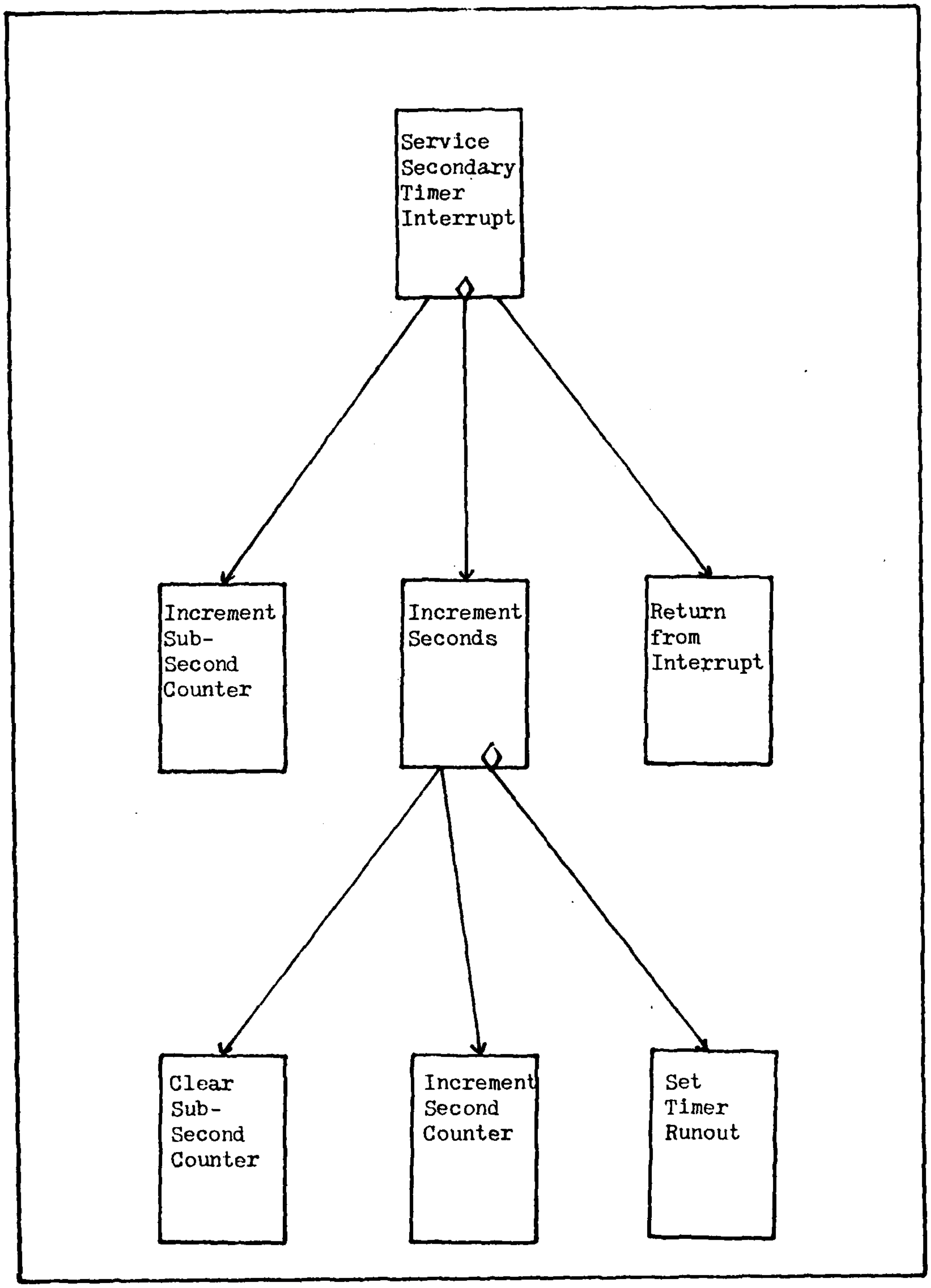
UTILITIES--Channel External Status Interrupt Routine



UTILITIES--Monitor Mode Channel Receive Interrupt Service Routine



UTILITIES--Simulate Mode Channel Transmit Interrupt Service Routine



UTILITIES--Secondary Timer Interrupt Service Routine

Appendix D

Line Tester Users Manual

This appendix contains the users manual for use of the line tester software developed in this report.

USERS MANUAL

FOR

ASYNCHRONOUS

DATA COMMUNICATIONS

LINE TESTER

Contents

Page

I. Introduction 155

Invoking Execution 155
Mostek Z-80 Memory Configuration 158

II. Commands 159

Quick Reference of Commands 161

III. Commmand Definitions 162

Display 162
Histogram 162
LOGAB 163
LOGBR 164
LOGCC 164
LOGCE 165
LOGCL 165
LOGFP 166
LOGSM 166
Monitor 167
Paramter 167
Quit 168
Simulate 168

I. Introduction

The line tester software resides in programmable read only memory (PROM). The variables which it uses during execution are located in random access memory (RAM). The stack pointer is located in static RAM which is on the CPU card. A block diagram of the memory configuration is included at the end of this section.

The software operates in two major modes. One mode is the monitor mode in which the communications line is strictly monitored by receiving the data coming across the line with no transmission of data. The other mode is the simulate mode in which data is transmitted over the communications line and with the help of a loop back device the same data sent is then received for comparision tests. Both modes are controlled by a system level driver which is the entry point upon power-up and reset.

Invoking Execution

Invoking the execution of the software is therefore a matter of power-up and reset. The software will wait for a response from the console (any key) after executing its boot up sequence. For example, after powerup and reset the following will be displayed after any console key is depressed.

```
SYSTEM BOOTED
$
```

followed by the list of system level commands. Selection of the monitor mode will invoke the output of

```
MONITOR MODE
%
```

followed by the list of monitor mode commands. In like fashion

```
SIMULATE MODE
>
```

will be output followed by its command, if the simulate mode is selected.

A third mode of operation contained in the line tester software is the line parameter mode which supports both the monitor and simulate modes. The line parameter mode is entered to display current line parameter settings (those which the operator can change) or set new line parameters depending upon the type of communications line being interfaced with the SIO. Setting of line parameters only pertains to the SIO whereas the console interface is fixed at a common setting by the line tester software. The line parameter sequence may be entered from any level within the line tester software. For example, selecting the line parameter function will output

```
RESPOND TO THE FOLLOWING SEQUENCE OF QUESTIONS
TO SET LINE PARAMETER VALUES
*
```

The parameters which the operator can change are channel to be used, baud rate factor, parity setting, stop bits, and receive and transmit bits per character settings. The baud rate factor applies to the strapping of the baud rate jumpers on the SIO card. Currently, the baud rate for channel A has been strapped for 9600 Baud while channel B has been strapped for 19,200 Baud. These baud rates apply at a X16 baud rate factor (the fastest available in asynchronous transmission).

If in doubt as to what response is needed for any level and the menu for that level is not present, a carriage return will act as a command to display the menu.

An erroneous input will cause the computer initiated prompt to be displayed again along with a question mark.

Mostek Z-80 Memory Configuration

Starting and Ending Adresses in Hex	Contents
FF00-FFFF	Stack Pointer (256X8 Static RAM)
F000-FEFF	Not Used
E800-EFFF	Not Used
E000-E7FF	User Program (2K PROM)
D800-DFFF	User Program (2K PROM)
D000-D7FF	User Program (2K PROM)
C800-CFFF	User Program (2K PROM)
C000-C7FF	User Program (2K PROM)
B000-BFFF	Not Used
A000-AFFF	Not Used
9000-9FFF	Not Used
8000-8FFF	Not Used
7800-7FFF	User Program Variables (2K RAM)
7000-77FF	Available (2K RAM)
6000-6FFF	Available (4K RAM)
5000-5FFF	Available (4K RAM)
4000-4FFF	Available (4K RAM)
3000-3FFF	Available (4K RAM)
2000-2FFF	Available (4K RAM)
1000-1FFF	Available (4K RAM)
0000-0FFF	Available (4K RAM)

II. Commands

The commands available are either one character or 5 characters in length. The multi-character commands are used to begin tests. NO paramters follow any of the commands.

Each test or mode is initiated by one of the commands shown in the succeeding table. Responses to the questions following these commands may be either another command or a computer initiated response as prompted by the software. The computer initiated dialogue will direct you through the questions.

To further detail the commands, let's go through the commands to execute a monitor mode test.

Step 1. Turn on power, reset micro
Step 2. Depress any key on console <CR>
Step 3. Set line parameters <P>
Step 4. Select mode from system menu <M>
Step 5. Select test to be performed <LOGFP>
Step 6. Enter duration of test <002000>

The above sequence will execute the monitor mode framing and parity test for 20 minutes. After Step 6 the software will prompt the user that a test is underway. A TEST COMPLETED message will be output once the test has finished, and the error totals will also be output automatically. If the line paramters had already been set from a previous test, step 3 need not have been executed. Both channels must be initiated for the monitor mode tests while the simulate mode requires only one.

The prompt for time input will be in the format of HHMMSS where

HH = hours

MM = minutes

SS = seconds

The histogram was not developed however all error totals are output in decimal values by the hour in which the errors occured. Due to the screen limitations of the console, 4 rows are used to output the totals for each channel. Six hours are represented by each row. Nine digits represent each hour. For example, row 1 begins with the totals of the first hour and ends with the totals of the sixth hour. Row 2 begins with hour 7 and ends with hour 12 and so on.

Additionally, for the monitor or simulate mode tests which are timed, a number will be output to the operator console indicating which timer is going. This also serves to indicate to the operator that a test is underway. A numeric '1' will be output by the main timer while a numeric '2' will be output by the secondary timer.

Quick Reference of Commands

Command Abbreviation	Name
M	Monitor (Prompt = %)
S	Simulate (Prompt = >)
D	Display
P	Parameter (Prompt = *)
Q	Quit
H	Histogram
LOGFP	Framing and Parity Test
LOGCL	Carrier Loss Test
LOGCC	Character Count Test
LOGBR	Bit Rate Test
LOGAB	Alternating Bit Rate Test
LOGSM	Standard Message Test
LOGCE	CPU Emulation Test

III. Command Definitions

In explaining the format and function of each command a few conventions will be followed. The input by the operator will be enclosed in inequality symbols <> while the computer response will be capitalized and unmarked. The abbreviated form of the command is shown enclosed in parenthesis. The initialization of the SIO channels is accomplished by executing the line parameters sequence--command P.

DISPLAY (D)

FUNCTION:

This command will display the menu of commands of the current operating level.

HISTOGRAM (H)

FUNCTION:

This command will cause a menu of tests to appear on the console. The menu will differ depending upon which mode you are currently operating. Selection of one of the tests will cause that tests current error totals to be displayed on the screen. This command will not display a histogram of the error totals since that function was not designed.

The menus which will be displayed for the two modes of operation are as follows:

MONITOR MODE

A. FRAMING AND PARITY TEST TOTALS

B. CHARACTER COUNT TEST TOTALS

C. CARRIER LOSS TEST TOTALS

D. QUIT

SIMULATE MODE

A. BIT RATE TEST TOTALS

B. ALTERNATING BIT RATE TEST TOTALS

C. QUIT

The operator selects the test to be displayed by entering the corresponding letter.

LOGAB

FUNCTION:

This command will cause the execution of the simulate mode alternating bit rate test. This test sends a 512-byte buffer of data; receives it and compares for mismatches between what was sent and what was received. A relative time input is required along with the channel selection. For example, beginning from within the simulate mode

```
<LOGAB>
SELECT CHANNEL TO BE USED IN SIMULATION TEST
```

```
A. CHANNEL A
B. CHANNEL B
<A>
ENTER LENGTH OF TEST PERIOD.  FORMAT=HHMMSS
<010000>
ALTERNATING BIT RATE TEST
```

The above sequence will execute the test for 1 hour on channel A. Upon completion the error totals will be displayed.

LOGBR

FUNCTION:

This command will cause the execution of the simulate mode bit rate test. This test sends a 512-byte buffer of data; receives it and compares for mismatches between what was sent and what was received. A relative time setting is required along with the channel used to perform the test.

LOGCC

FUNCTION:

This command will cause the execution of the monitor mode character count test. This test will count all characters coming across the line being monitored. Both the 'transmit' and the 'receive' lines will be monitored. A relative time input is required. Both SIO channels must be initialized. For example, beginning within the monitor mode

```
<LOGCC>
ENTER LENGTH OF TEST PERIOD.  FORMAT=HHMMSS.
<020000>
CHARACTER COUNT TEST
```

The above sequence will execute the character count test for 2 hours.

LOGCE

FUNCTION:

This command will cause the execution of the simulate mode CPU emulation test. This test will prompt the operator for a character, and after the character is input it will be transmitted through the SIO, over the communications line, looped back through the SIO and displayed back on the user console.

LOGCL

FUNCTION:

This command will cause the execution of the monitor mode carrier loss test. Both channels of the SIO must be initiated however only channel A's carrier signal is monitored. A relative time duration is required for this test along with the duration of carrier loss before being counted as an error. A test for the carrier signal is performed first. If no signal is found after 1 minute, the test is aborted.

LOGFP

FUNCTION:

This command will cause the execution of the monitor mode framing and parity test. Both channels must be initiated. A relative time input is required. This test will log the occurence of either a parity error or a framing error as one error.

LOGSM

FUNCTION:

This test will cause the execution of the simulate mode standard message test. After inputting this command, a menu of standard message tests will appear. The user may execute standard message tests four different ways. One way is to input the message to be transmitted over the line and displayed back on the user console and repeat the operation manually. A second way is to input the message and also the relative time the message is to be automatically repeated. The third and fourth ways are to send the resident message with manual repeat or automatic repeat. The menu is shown below.

SELECT ONE

A. MSG TO BE INPUT AND REPEATED BY OPERATOR

B. MSG TO BE INPUT BY OPERATOR WITH AUTO REPEAT

C. RESIDENT MSG REPEATED BY OPERATOR

D. RESIDENT MSG WITH AUTO REPEAT

E. QUIT

The operator performs a standard message test by selecting the corresponding letter.

MONITOR (M)

FUNCTION:

This command will cause the line tester to enter the monitor mode. The tests available in the monitor mode include

LOGFP

LOGCL

LOGCC

Both channels of the SIO must be initialized in the monitor mode. This mode only receives data through the SIO and no data is transmitted.

PARAMETER (P)

FUNCTION:

This command will cause the line tester to enter the line parameter sequence whereby the following may be set

CHANNEL
BAUD RATE FACTOR
STOP BITS
PARITY SETTING
RECEIVE BITS PER CHARACTER
TRANSMIT BITS PER CHARACTER

These settings apply to the SIO channels A and B.

QUIT (Q)

FUNCTION:

This command will terminate the current level of operation. The level transferred to will be the one in which the current level was called.

SIMULATE (S)

FUNCTION:

This command will cause the line tester to enter the simulate mode. A menu of commands will be displayed which may be performed in the simulate mode. Only the channel which is to be used in the simulate test need be initialized via the line parameter sequence. The simulate tests transmit and receive data through the SIO. The tests available include

LOGBR

LOGAB

LOGSM

LOGCE

A relative time input is required for the first three tests listed.

Appendix E

Programming the SIO

This appendix discusses the details of how to properly initialize and use the Z-80 serial input/output (SIO) device. There exist procedures that must be followed for proper SIO initialization. A good reference on the SIO can be found in the book entitled, An Introduction to Microcomputers: Volume 3 Some Real Support Devices by Jerry Kane. This book is published by Adam Osborne and Associates. In addition, Zilog has a clear and understandble application note on asynchronous communications using the Z-80 SIO which would be very helpful in understanding the SIO. These references are strongly recommended for a thorough understanding of the complex SIO.

The Z-80 SIO is a dual channel serial I/O device which can transmit and/or receive serial data on each of its channels. Each channel can be independently programmed. In asynchronous communications, each fixed-length chararcter is preceded and terminated by framing bits to identify the beginning and ending of the character. Included with the framing bits are stop bits which indicate when the characters are transmitted.

There are three modes of operation on the SIO. They are polled, interrupt, and block transfer. The references discussed at the beginning of this appendix describe these modes rather well. The interrupt mode will be discussed here due to its relevance to the thesis topic.

Within the interrupt mode the SIO informs the CPU via an interrupt signal that a character transfer is required. The interrupt service routines are accessed via an interrupt table. The interrupt table will contain the addresses of each interrupt service routine. The SIO will provide the lower 8 bits of the address within the interrupt table while the Interrupt Register (I) of the CPU provides the upper 8 bits.

The SIO has two channels as previously mentioned. Addressing the SIO then is a matter of addressing these two channels. The port addresses are user defined and usually range from 0 to 255. Each channel is given two port assignments. One port assignment refers to the data port while the second assignment refers to the control port. The data port obviously is used to transfer data while the control port is used to initialize the SIO and obtain status flags.

There are normally 6 steps to be accomplished to properly initialize the SIO. The steps are listed below and explained in detail later. Initialization is accomplished by writing values to the control port.

Step 1. Reset channel

Step 2. Load write register 4 (mode register)

Step 3. Load write register 3 (receive register)

Step 4. Load write register 5 (transmit register)

Step 5. Load write register 2 (interrupt address)

Step 6. Load write register 1 (interrupt mode)

As indicated above there are many registers associated with each channel. In fact, each channel has 8 write registers and 3 read registers which are all accessed via the control port.

Accessing the registers is done by using a master write register (write register 0). To access any of the registers WR0 must first be written to with the value of the register desired. For example, to access WR2 the first step would be to write the value '2' to WR0. The next write to the control port will be to WR2. Upon completion of the write to WR2, WR0 will reset automatically to point to itself thereby readying itself for the next access.

The 6 steps listed above then are actually 12, since

WR0 must be used for each write to the other registers. WR0 not only contains a pointer to the register to be accessed but also has 3 bits designated as commands. One of the commands is the reset command. The reset command must be the only value issued to the channels control port when beginning initialization because it takes a few extra clock cycles to perform the reset. In other words do not point to another register other than WR0 when issuing a reset command.

In accomplishing the other steps, WR0 is used to point to the register desired and the value written to that register is user defined according to the requirements to be satisfied by the SIO. The references adequately detail the numerous options. The interrupt mode register (WR1) and the interrupt address register (WR2) need only be initialized if the SIO is to operate in the interrupt mode. If the interrupt mode is chosen then of course interrupt service routines must be generated to accomodate the interrupts. The SIO can be programmed to modify its portion of the interrupt table address, according to the type of error which may have occured or normal reception and transmission interrupts. Both channels are capable of generating interrupts.

The read registers are used to obtain status flags. The statuses are triple buffered and obtained by again using WR0 to point to the register desired. This time instead of outputting to the port an input is performed thereby reading the read registers.

In conclusion the SIO is very complex and it is strongly advised that the references listed in the beginning be obtained to properly use the SIO. As a general note most of the problems associated with using the SIO have been because of improper initialization.

Vita

Captain David L. Hartmann was born on October 18, 1953 in Blanco, Texas. In 1972 he graduated from Fredericksburg Senior High School in Fredericksburg, Texas. He attended Southwest Texas State University from which he received a Bachelor of Science degree with a major in Computer Science in 1976. Following graduation he was assigned to the 4602 Computer Services Squadron located at Peterson AFB, Colorado as a system analyst/programmer from 1976 to 1980. He entered the Air Force Institute of Technology in June of 1980.

Permanent address: 308 E. Travis Street
Fredericksburg, Texas

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER AFIT/GCS/EE/81D-11	2. GOVT ACCESSION NO. AD-A115 560	3. RECIPIENT'S CATALOG NUMBER

4. TITLE (*and Subtitle*) DEVELOPMENT OF A MICROPROCESSOR-BASED ASYNCHRONOUS DATA COMMUNICATIONS LINE TESTER	5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(*s*) David L. Hartmann, Captain, USAF	8. CONTRACT OR GRANT NUMBER(*s*)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Weapons Laboratory (AFWL/ADD) Computation Division, Bldg 410 Kirtland AFB, New Mexico 87107	12. REPORT DATE December, 1981 13. NUMBER OF PAGES 184
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*)	15. SECURITY CLASS. (*of this report*) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

15 APR 1982

Lyn E. Wolaver

Dean for Research and
Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

FREDERIC C. LYNCH, ~~Major,~~ USAF
Director of Public Affairs

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Microprocessor
Asynchronous Communications
Data Line Tester
Communications Software
Structured Programming

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

See reverse

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

20. ABSTRACT

Software was developed to support a microprocessor-based asynchronous data communications line tester which operated on digital data. The software functions included a monitor mode and a simulate mode both of which could be executed through interactive dialogue with the operator of the system. The software was to reside in PROM and execute upon power up and reset.

A structured top down approach was used to design a transaction oriented software line tester. The design was implemented on a Mostek MDX microcomputer system which included a serial input/output (SIO) interface, a Z-80 CPU module, 32K dynamic RAM module, and an EPROM/UART module. Data line testing was performed through the SIO utilizing its interrupt mode and status gathering capabilities to capture error conditions. The software, written in Z-80 assembly language, was tested using both "white box" and "black box" testing strategies.

The microprocessor-based line tester proved to be versatile and efficient in performing asynchronous data communications line testing. Designing the software using the top down approach was effective in permitting structure changes. The large and powerful Z-80 instruction set provided a great degree of versatility for performing the line tester functions.

ATE

LMED

8